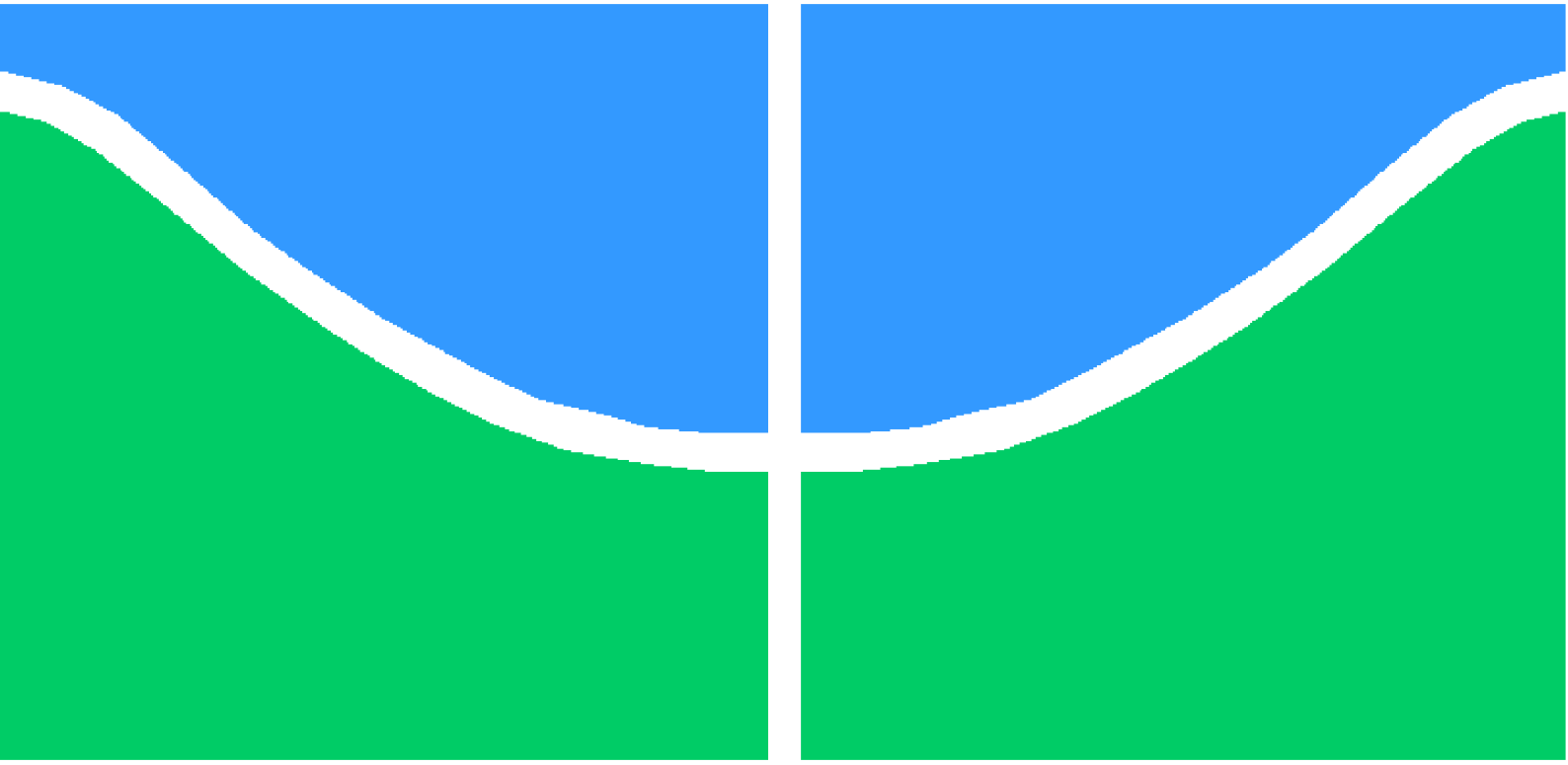


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia Eletrônica

Arquitetura de Hardware dedicada para o cálculo da distância frontal baseado em segmentação por cores usando visão estéreo aplicada em robótica móvel

Autor: Thales da Cruz Portela

Orientador: Prof. Dr. Daniel Maurício Muñoz Arboleda

Brasília, DF

2014

Thales da Cruz Portela

Arquitetura de Hardware dedicada para o cálculo da distância frontal baseado em segmentação por cores usando visão estéreo aplicada em robótica móvel

Monografia submetida ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Daniel Maurício Muñoz Arboleda

Brasília, DF

2014

Thales da Cruz Portela

Arquitetura de Hardware dedicada para o cálculo da distância frontal baseado em segmentação por cores usando visão estéreo aplicada em robótica móvel/ Thales da Cruz Portela. – Brasília, DF, 2014-

83 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Daniel Maurício Muñoz Arboleda

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2014.

1. Palavra-chave01. 2. Palavra-chave02. I. Prof. Dr. Daniel Maurício Muñoz Arboleda. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Arquitetura de Hardware dedicada para o cálculo da distância frontal baseado em segmentação por cores usando visão estéreo aplicada em robótica móvel

CDU 02:141:005.6

Thales da Cruz Portela

Arquitetura de Hardware dedicada para o cálculo da distância frontal baseado em segmentação por cores usando visão estéreo aplicada em robótica móvel

Monografia submetida ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Trabalho aprovado. Brasília, DF, 18 de novembro de 2014:

Prof. Dr. Daniel Maurício Muñoz Arboleda
Orientador

Prof. Me. Renan Utida Ferreira
Convidado 1

Prof. Dr. Gilmar Silva Beserra
Convidado 2

Brasília, DF
2014

Dedico este trabalho aos meus pais, João e Rita, que sempre me apoioram e me deram todo o suporte necessário durante a minha formação.

Agradecimentos

Agradeço à minha família pelo incentivo e auxílio durante todo o curso de engenharia eletrônica como também durante toda minha formação anterior.

Agradeço a todos os meus amigos que de alguma forma estiveram presentes e me ajudaram durante o período do meu curso.

Agradeço a todos os meus colegas de curso que junto comigo passaram por várias noites mal dormidas e compartilharam os momentos de tensão em projetos e avaliações durante esses cinco anos.

Agradeço a todos os meus professores que lecionaram alguma disciplina ou que tive a oportunidade de desenvolver projetos em conjunto, pela passagem do conhecimento, por compartilhar experiências e principalmente por sempre me motivar e ser importantes referências.

Agradeço ao Camilo Sánchez Ferreira e ao Antônio Pedro Cardillo Bittencourt por generosamente cederem materiais próprios que embasaram o desenvolvimento desse trabalho. Agradeço também ao Luis Contreras por gentilmente me instruir e responder dúvidas relativas a uma das plataformas utilizadas no projeto.

Agradeço ao meu professor orientador Daniel Maurício Muñoz Arboleda pela paciência, disponibilidade, dedicação e por me ajudar a desenvolver esse trabalho.

"O homem que vê o mundo aos 50 anos
do mesmo modo que via aos 20,
perdeu 30 anos da sua vida."
(Muhammad Ali)

Resumo

O presente trabalho apresenta uma proposta de arquitetura de *hardware* dedicada para o cálculo da distância frontal a um objeto de interesse através do uso de um sistema de visão estéreo de duas câmeras. Adicionalmente, o sistema desenvolvido foi acoplado a uma plataforma robótica móvel a fim de detectar e seguir o objeto a uma distância constante. O sistema de visão estéreo proposto foi desenvolvido com base em uma arquitetura de *hardware* dedicada de segmentação por cores desenvolvida previamente, a qual identifica um objeto colorido em uma tonalidade específica e calcula o centro geométrico do mesmo. A arquitetura de segmentação por cores foi modificada agregando a funcionalidade de se escolher a cor do objeto de interesse pelo usuário. A arquitetura de segmentação por cores foi implementada no kit de desenvolvimento DE2 provido do FPGA *Cyclone II* EP2C35 da Altera e faz uso de um kit de câmera digital de 5 *Megapixel* TRDB_D5M e de um *display* LCD *touch screen* TRDB_LTM, ambos da *Terasisc Technologies*. A arquitetura de cálculo da distância frontal, por sua vez, usa dois módulos da câmera digital especificada acima e foi desenvolvido no kit de desenvolvimento DE2-70; uma versão modificada da plataforma DE2, porém com um FPGA *Cyclone II* EP2C70 e mais memória; e foi integrada posteriormente ao robô *Pioneer 3-AT* da *Adept MobileRobots*. O sistema modificado de segmentação por cores foi validado para o grupo das seis cores primárias e secundárias "puras", observou-se que a cor vermelha apresentou melhor resultado e, portanto, foi utilizada no sistema integrado à plataforma robótica móvel. Os testes da arquitetura de cálculo da distância frontal foram realizados para as distâncias de $1m$; $1,5m$; $2m$; $2,5m$ e $3m$ que chegaram a apresentar erros relativos da ordem de 1% e até menores. Os testes do sistema integrado à plataforma conseguiram controlar o movimento do robô de forma adequada e alcançaram os melhores resultados para a distância de $1m$.

Palavras-chaves: *Hardware* Reconfigurável. FPGA. Visão Estéreo. Espaço de cores HSV. Visão robótica. Robótica Móvel. Sistemas Embarcados.

Abstract

This work proposes a hardware architecture for computing the distance between an object of interest and a stereo vision system composed of two cameras. In addition, this system was attached to a mobile robotic platform in order to track an object at a predefined distance. Such stereo vision system was based on a previous implementation of a color segmentation hardware architecture capable of identifying a specific colored object and compute its geometric center. The color segmentation module was then modified in order to add a function of choosing the object's color to be pursued through an additional input provided by the user. The color segmentation architecture was implemented using the Altera DE2 development kit which contains an Altera Cyclone II EP2C35, a TRDB_D5M 5 Mega pixel digital camera development kit and an LCD Touch Panel Module (LTM) board, provided by Terasic Technologies. In other hand, the architecture for computing the front distance is going to use two TRDB_D5M 5 Mega pixel digital camera development kit and was implemented using an Altera DE2-70 board, which is a modified version of DE2 but featuring an Altera Cyclone II EP2C70 and additional memory instead, and then was integrated to a Pioneer 3-AT robot provided by Adept MobileRobots. The modified color segmentation system was validated for a 6 colors group composed of the primary and secondary colors. Experiments have demonstrated that red color achieved best segmentation results and, therefore, it was used in the integration with the mobile robotic platform. Tests on front distance module were done for $1m$, $1.5m$, $2m$, 2.5 and $3m$ lengths and presented relative errors close and below 1%. Final system applied to the mobile robotic platform was able to control robot's motion satisfactorily and achieved best results for $1m$ length.

Key-words: Reconfigurable Hardware. FPGA. Stereo Vision. HSV Space Color. Robot Vision. Mobile Robotics. Embedded Systems.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

3D	*Three-Dimensional*
ASIC	*Application-Specific Integrated Circuit*
CI	*Circuito Integrado*
DE	*Development and Educational*
DSP	*Digital Signal Processor*
FPGA	*Field-Programmable Gate Array*
GRACO	Grupo de Automação e Controle
HDL	*Hardware Description Language*
HSB	*Hue, Saturation and Brightness*
HSI	*Hue, Saturation and Intensity*
HSL	*Hue, Saturation and Lightness*
HSV	*Hue, Saturation and Value*
I/O	*Input/Output*
ISE	*Integrated Software Environment*
IrDA	*Infrared Data Association*
LCD	*Liquid-Crystal Display*
LED	*Light Emitting Diode*
LUT	*Look-Up Table*
MSE	*Mean Squared Error*
NTSC	*National Television System Committee*
PAL	*Phase Alternating Line*
PID	Proporcional Integral Derivativo
PLL	*Phase-Locked Loop*

RGB	*Red, Green and Blue*
SBT	*Software Build Tools*
SD	*Secure Digital*
SDRAM	*Synchronous Dynamic Random Access Memory*
SRAM	*Static Random Access Memory*
SSRAM	*Synchronous Static Random Access Memory*
UnB	Universidade de Brasília
ULA	Unidade Lógica Aritmética
USB	*Universal Serial Bus*
VHDL	*VHSIC Hardware Description Language*
VGA	*Video Graphics Array*
VHSIC	*Very-High Speed Integrated Circuit*

Sumário

1 Introdução

A robótica atingiu seu auge de sucesso na fabricação industrial com o uso de braços robóticos, manipuladores, que desempenhavam uma função específica em uma linha de montagem. Porém, para atingir seu pleno desempenho, era necessário mobilidade a fim de agregar flexibilidade para realizar uma determinada tarefa de forma mais efetiva (SIEGWART; NOURBAKHSH, 2004). Juntamente com a mobilidade, a visão em robôs os qualifica a uma gama de aplicações como navegação, servovisão para rastreamento e manipulação de objetos, reconhecimento e categorização de objetos, segurança e tomadas de decisão de alto nível (UDE, 2010).

Similarmente ao que aconteceu com os computadores *mainframes* que evoluíram para os computadores pessoais e dispositivos portáveis, a robótica móvel passou por um processo onde robôs eram controlados por sistemas computacionais grandes, pesados e caros que tinham que ser ligados à plataforma robótica para sistemas embarcados leves, pequenos e com custo reduzido que podem ser acoplados fisicamente ao robô (BRAUNL, 2003).

Um sistema embarcado é um sistema computacional embutido em um produto ou componente, logo este sistema é projetado para realizar uma tarefa ou um conjunto de tarefas específicas (BAIELY, 2011). Sistemas embarcados possuem capacidades de sensoriamento e, dependendo da aplicação, podem ter restrições de consumo de potência, portabilidade e alto desempenho computacional (GARCIA et al., 2006).

Arranjo de portas programável em campo, do inglês *Field-Programmable Gate Arrays* (FPGAs), são dispositivos integrados usados na implementação de circuitos digitais através de um processo de configuração ou programação (FERNANDEZ et al., 2004). O papel dos dispositivos FPGAs evoluiu com o passar dos anos, deixando de ser usado para substituir CI (Circuito Integrado) de pequena e média escala até ser um componente com crescente importância no campo de sistemas embarcados (SASS; SCHMIDT, 2010). Em geral, os dispositivos FPGAs oferecem uma grande vantagem para os projetistas de sistemas embarcados principalmente por oferecer um enorme grau de flexibilidade, fato que é bastante interessante para atingir os requisitos complexos e exigentes dos sistemas embarcados atuais (SASS; SCHMIDT, 2010).

A visão computacional é dada por um conjunto de algoritmos que buscam obter informações contidas em imagens digitais (ESTEVES; FEITOSA; FERNANDES, 2012). Virtualmente todos os algoritmos de processamento de imagens são compostos por sequencias de operações, o que gera o chamado paralelismo temporal (BAIELY, 2011). Para um determinado nível de algoritmo, uma grande quantidade de paralelismo é encon-

trada na forma de *loops*. No processamento de imagens, várias operações são repetidas para cada *pixel* da imagem através de um *loop*, conceito conhecido como paralelismo espacial (BAIELY, 2011). O paralelismo lógico corresponde aos *loops* mais internos do código e equivalem ao reuso de blocos lógicos para realizar uma mesma operação (BAIELY, 2011). Sabendo que os FPGAs possuem uma grande variedade de dispositivos lógicos programáveis que podem ser usados em paralelo, o uso dessas plataformas inerentemente paralelas para implementação de algoritmos de processamento de imagens torna-se uma solução vantajosa para transformar as propriedades desses algoritmos em melhorias no desempenho computacional da aplicação.

1.1 Justificativa

O objetivo de aplicar algoritmos de visão computacional em robótica móvel é tornar possível ao robô a percepção de um mundo externo a fim de desempenhar uma grande variedade de tarefas (UDE, 2010). Técnicas visuais para detecção e rastreamento de elementos significativos em uma cena, chamados de características, tem sido melhoradas extensivamente durantes os últimos anos e usadas para propósitos de localização e/ou navegação (UDE, 2010).

A visão estéreo utiliza duas ou mais câmeras direcionadas para uma mesma cena. A partir das correspondências entre pontos das imagens geradas e do conhecimento da geometria e das propriedades das câmeras, é possível determinar a profundidade dos objetos em uma cena. A estereoscopia visual pode ser empregada para solucionar diversos problemas como o mapeamento de ambientes, cálculos de distâncias e reconhecimentos de objetos tridimensionais (ESTEVES; FEITOSA; FERNANDES, 2012).

O sistema HSV utiliza as noções comuns de tonalidade, saturação e valor como as três componentes que descrevem uma cor (SMITH, 1978). O espaço HSV tem como vantagem não só ser muito mais intuitivo do que o sistema RGB, como também possui a propriedade da conversão entre o espaço RGB para o HSV ser bastante fácil de se fazer, não exigindo grande quantidade de recursos computacionais para a transformação de espaços (SMITH, 1978).

Algoritmos de processamento de imagens mostram propriedades paralelas em diversas frentes devido ao fato de serem feitas várias etapas de processamento entre a imagem adquirida e o resultado final, devido à propriedade espacial dos *pixels* da imagem e devido ao reuso de blocos menores durante o algoritmo. A partir disso, pode-se fazer uso de plataformas reconfiguráveis e, uma vez que FPGAs modernos têm recursos suficientes para possibilitar aplicações inteiras serem desenvolvidas em um único FPGA, estes tornam-se uma escolha ideal para sistemas de visão em tempo real (BAIELY, 2011).

Com base na extensa lista de aplicações possíveis em torno da visão computacional

aplicada à robótica móvel e nas vantagens que o uso de dispositivos reconfiguráveis como o FPGA agregam aos algoritmos que apresentam paralelismo intrínseco, o presente trabalho tem sua motivação.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral do trabalho é desenvolver uma arquitetura de *hardware* dedicada para o cálculo da distância entre um sistema de visão estereoscópica de duas câmeras e um objeto de interesse através do cálculo da disparidade de um ponto no espaço visto nas imagens provenientes das câmeras. Adicionalmente, o sistema de visão estéreo desenvolvido em plataforma FPGA será acoplado a um robô móvel que deverá seguir o objeto de interesse a uma distância constante de acordo com a informação recebida.

1.2.2 Objetivos Específicos

- A partir de uma solução de segmentação por cores em plataforma FPGA (BITTENCOURT, 2012) usada como ponto de partida do presente trabalho, modificar, validar e analisar a implementação a fim de agregar a possibilidade de escolha pelo usuário da cor a ser segmentada, inserida como entrada externa do sistema embarcado.
- Duplicar, validar e analisar a arquitetura de segmentação por cores resultante do item anterior na plataforma FPGA, possibilitando a inserção da segunda câmera no sistema embarcado.
- Desenvolver, validar e analisar o *hardware* dedicado ao cálculo das disparidades e determinar a distância frontal entre o sistema de visão estéreo e o objeto de interesse.
- Validar a solução desenvolvida através do acoplamento da plataforma FPGA em um robô móvel para rastreamento do objeto de interesse.

1.3 Aspectos Metodológicos

O presente trabalho pode ser dividido nas seguintes etapas no que se refere à metodologia utilizada:

- A primeira etapa do trabalho foi constituída por uma revisão detalhada dos conceitos envolvidos no escopo do projeto. Dentre os temas pesquisados estão a conversão do espaço de cores RGB para o HSV e suas vantagens e desvantagens; a estrutura das plataformas de *hardware* reconfigurável e suas vantagens no projeto de sistemas

embarcados e na implementação de algoritmos com paralelismo intrínseco; a visão estereoscópica e o consequente cálculo da distância frontal através das disparidades entre as imagens; e aplicações de visão computacional voltadas à robótica móvel.

- Tendo em vista que este trabalho utiliza como ponto de partida o processo de segmentação por cores desenvolvido em (BITTENCOURT, 2012), a segunda etapa é a análise e validação das arquiteturas previamente implementadas. Nesta etapa foi usada a metodologia *top-down* para análise do sistema.

- A fase 3 é baseada na metodologia *bottom-up* para desenvolvimento e validação das implementações em *hardware* de forma hierárquica. Nesta fase, a modificação da arquitetura de segmentação por cores foi realizada a fim de se adicionar a funcionalidade de escolha da cor do objeto a ser segmentado. A partir do código HDL resultante, foi feita a duplicação da arquitetura de segmentação de imagens e implementado os módulos do cálculo das disparidades e da determinação da distância frontal.

- Na fase 4 foi realizada a caracterização da solução de visão estéreo proposta no que diz respeito aos resultados da síntese lógica gerados pela ferramenta automática de síntese, o consumo de recursos e a frequência máxima de operação. Nesta etapa também foi realizada a validação das implementações através de testes do sistema de visão estéreo tanto para a correta segmentação da cor indicada como, posteriormente, para a distância frontal computada. O cálculo do número de acertos e uma análise de erros foram realizados a fim de validar ambos resultados, respectivamente.

- A fase 5 foi dedicada à integração do sistema embarcado de visão estéreo implementado na plataforma FPGA com o robô móvel *Pioneer 3-AT*.

- A fase 6 foi responsável pela validação do sistema integrado através de testes em ambientes controlados tendo em vista que o presente trabalho se realizou no escopo acadêmico e que diversas situações do ambiente real ainda não foram levadas em consideração na implementação realizada.

- A fase 7 foi a fase de documentação e possíveis publicações do presente trabalho. Foi escrita a versão final do Trabalho de Conclusão de Curso em Engenharia Eletrônica na Universidade de Brasília e, a partir deste documento e dos resultados obtidos, será proposta a escrita de artigos em periódicos da área de robótica móvel, *hardware* reconfigurável, sistemas embarcados e áreas correlatadas.

1.4 Organização do Trabalho

O presente trabalho divide-se da seguinte maneira: o capítulo 2 apresenta a fundamentação teórica pertinente às áreas abordadas no trabalho, abrangendo os espaços de cores RGB e HSV, visão estéreo e o cálculo da profundidade de uma cena a partir duas imagens, e a estrutura e aspectos relevantes de *hardware* reconfigurável. O capítulo 3 explica as implementações desenvolvidas no decorrer do projeto. O capítulo 4 apresenta e discute os resultados obtidos ao final do trabalho. Por fim, o capítulo 5 destaca as conclusões e propostas de trabalhos futuros.

2 Fundamentação Teórica

2.1 Imagem Digital

Uma imagem pode ser definida como uma função bidimensional $f(x, y)$ onde x e y são as coordenadas espaciais e f é a intensidade ou valor de cinza em um determinado par coordenado, para imagens monocromáticas. Quando tanto as coordenadas espaciais quanto seu valor associado são quantidades discretas, a imagem é chamada de imagem digital. Assim, uma imagem digital é composta por uma quantidade finita de elementos, esses elementos são chamados de elementos da figura, elementos de imagem, *pels* ou *pixels* (GONZALEZ; WOODS, 2008).

Uma imagem capturada por um sensor precisa passar por duas etapas para ser processada por um computador: digitalização e quantização. Digitalização da imagem é quando a função $f(x, y)$ é amostrada em uma matriz de M linhas e N colunas e quantização é a atribuição de um valor inteiro a cada amostra, ou seja, a função da imagem é representada por uma quantidade limitada de valores(SONKA; HLAVAC; BOYLE, 2008).

A Fig. (1) ilustra os processos definidos acima. A Fig. (1 (a)) mostra uma imagem contínua a qual se quer converter em digital. A função de uma variável mostrada na Fig. (1 (b)) é o gráfico dos valores de intensidade da Fig. (1 (a)) sobre a linha AB. A Fig. (1 (c)) ilustra a amostragem da imagem na coordenada x. A linha AB é dividida em intervalos iguais ilustrados pelas marcas verticais na parte debaixo do gráfico e apenas nesses locais são adquiridos os valores de intensidade da imagem, representados pelos quadrados brancos. A Fig. (1 (d)) representa o último passo para a conversão digital. Nessa figura, os níveis de cinza que até então podem assumir valores contínuos são limitados em intervalos discretos ilustrados pelo eixo vertical da Fig. (1 (d)). Os valores contínuos são então convertidos para o valor discreto mais próximo, o que corresponde à quantização. A repetição desse processo linha-a-linha gerará uma imagem digital referente à versão contínua da Fig. (1 (d)) (GONZALEZ; WOODS, 2008).

2.2 Espaço de Cores

Um espaço de cores é basicamente um sistema de coordenadas e um subespaço onde cada ponto nele inserido representa uma determinada cor (GONZALEZ; WOODS, 2008).

Os espaços de cores existentes atualmente podem se dividir em geral em dois grupos: (a) orientados ao hardware e, (b) orientados à aplicação. No primeiro grupo, o

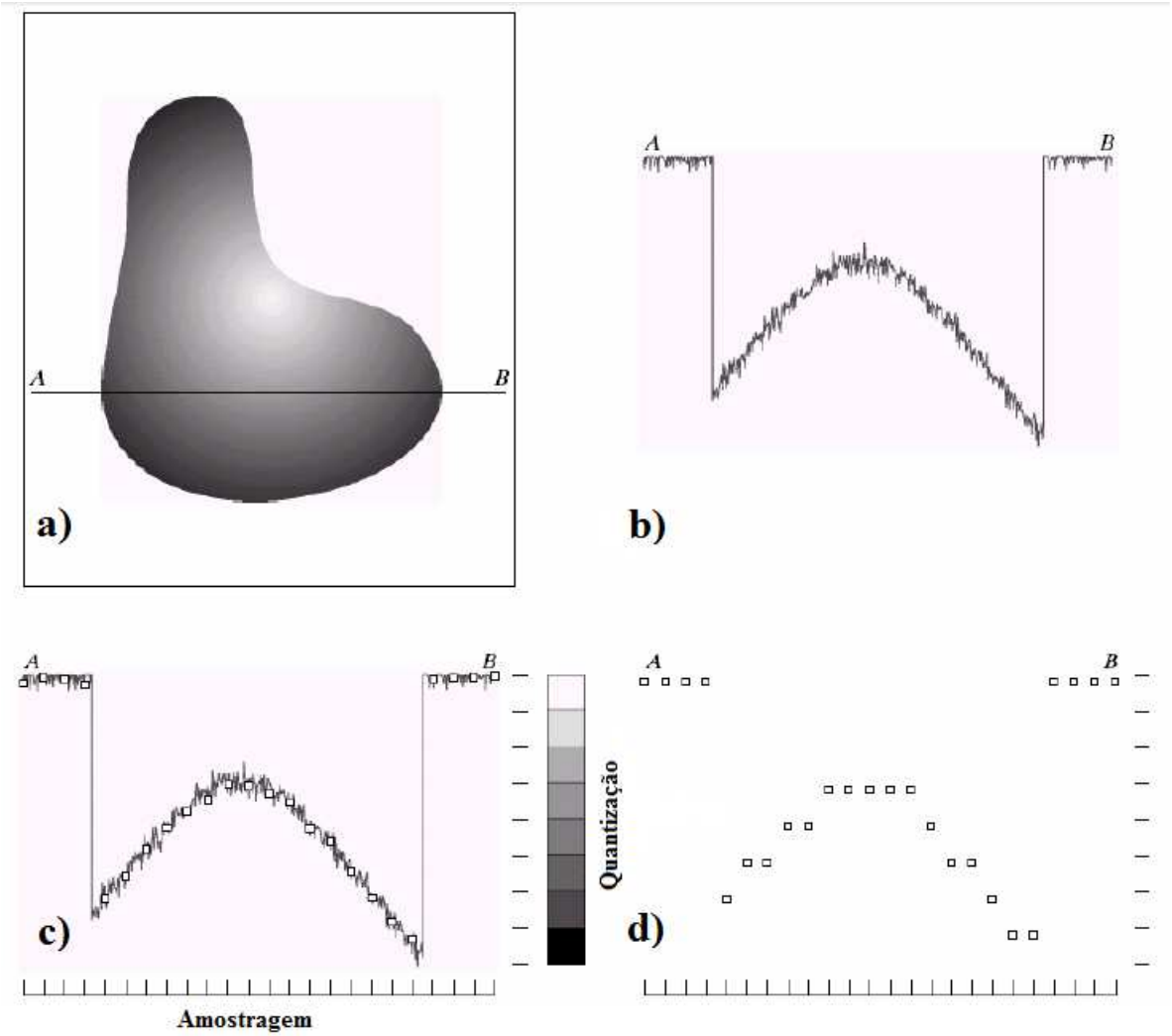


Figura 1 – Geração de uma imagem digital. (a) Imagem contínua. (b) Linha AB destacada na imagem para ilustrar os conceitos de amostragem e quantização. (c) Amostragem (d) Quantização e versão digital da linha AB. (Modificado de(GONZALEZ; WOODS, 2008)).

sistema de cores mais usado em processamento de imagens é o RGB; que é proveniente das palavras em inglês *red*, *green* e *blue* que significam os canais de cores vermelho, verde e azul; e que possui aplicação em monitores coloridos e para uma variedade de câmeras de vídeo (GONZALEZ; WOODS, 2008). No segundo grupo pode-se exemplificar o espaço HSV, cujas componentes vêm dos termos em inglês *hue*, *saturation* e *value* que significam tonalidade, saturação e valor, respectivamente. Na literatura, existem diversas variações desse sistema de cor e também da sua nomenclatura, assim, termos como HSB, HSI e HSL são comumente encontrados. A terceira componente de cada um desses espaços de cores se referem aos termos em inglês *brightness*, *intensity* e *lightness* e se significam brilho, intensidade e luminosidade, respectivamente. Neste segundo grupo, as componentes são mais intuitivas e voltadas à percepção humana das cores, logo é mais usado em aplicações onde a manipulação das cores é feita pelo usuário.

2.2.1 Espaço de Cores RGB

O espaço de cores RGB é o sistema de cores aditivo onde cada cor é obtida através da soma das componentes primárias vermelho, verde e azul. O modelo é apresentado graficamente por um sistema de coordenadas cartesianas 3D e um subespaço da forma de um cubo delimita as cores representadas (Fig. 2).

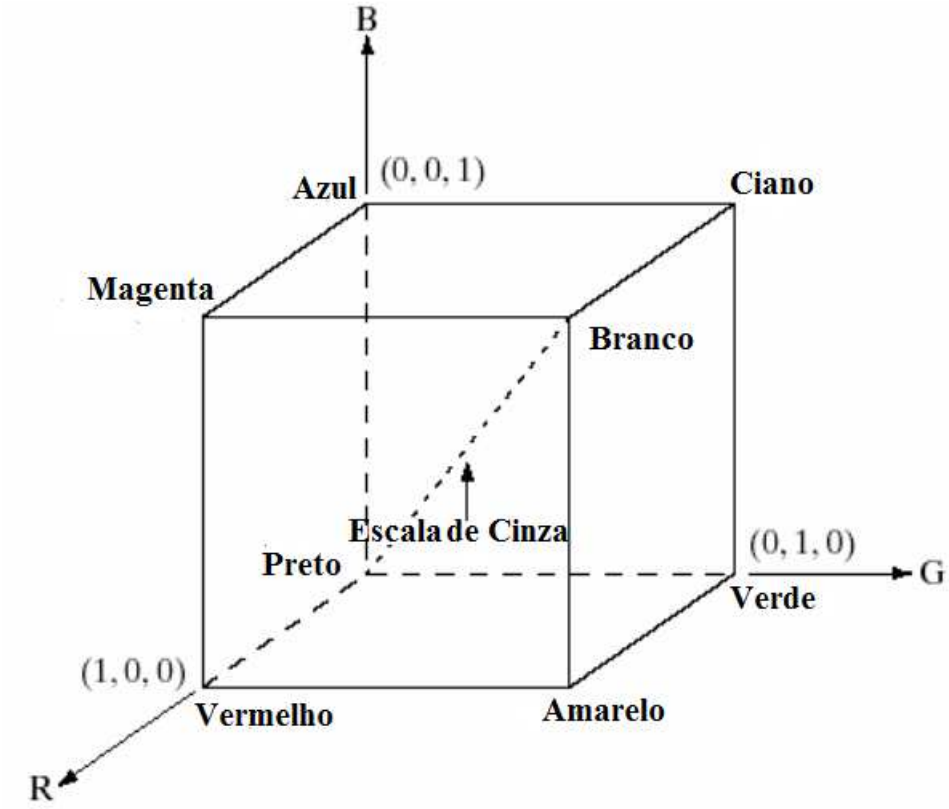


Figura 2 – Representação gráfica do espaço de cores em sistema de coordenadas cartesianas RGB. (Modificado de (GONZALEZ; WOODS, 2008)).

Na Fig. (2), os vértices do cubo sobre os eixos cartesianos representam as cores primárias vermelho, verde e azul; os outros vértices do cubo representam as cores secundárias magenta, ciano e amarelo; juntamente com as cores preta e branca, que estão posicionadas na origem e no ponto onde as coordenadas são máximas, respectivamente. É interessante notar que a reta determinada pelos pontos das cores branca e preta, que é também os pontos onde todas as componentes possuem o mesmo valor, nada mais é do que a escala de cinza.

2.2.2 Espaço de Cores HSV

O fato da adição das componentes do espaço RGB não gerar uma resultante intuitiva à percepção humana faz com que o seu uso não seja adequado para algumas aplicações (JOBLOVE; GREENBERG, 1978). Uma opção é a utilização do espaço de coordenadas

cilíndricas HSV. H está ligado à matiz ou tonalidade, S se refere à saturação e V vem do termo "valor"e está relacionado ao brilho. A Fig. (3) mostra a representação gráfica 3D do espaço de cores cilíndrico HSV juntamente com três diferentes vistas da mesma, que dão um melhor entendimento do comportamento das cores para variações dos componentes H, S e V.

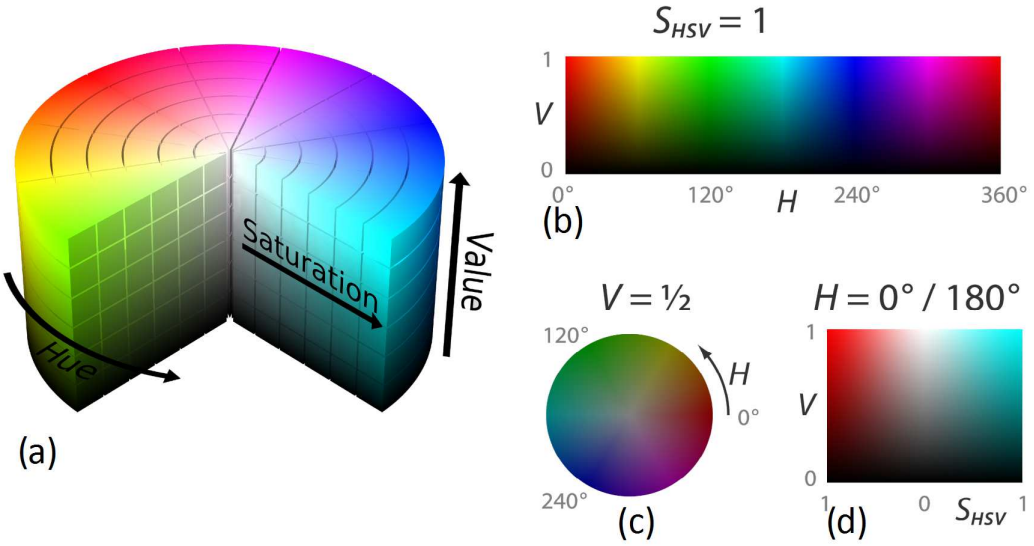


Figura 3 – Representação gráfica do espaço de cores em sistema de coordenadas cilíndricas HSV e três diferentes vistas do respectivo cilindro. (a) Espaço de cores 3D HSV. (b) Vista da área lateral do cilindro (Componente de saturação igual a 1). (c) Vista superior de um corte transversal realizado no ponto em que a componente de valor é igual $\frac{1}{2}$. (d) Vista frontal de um corte longitudinal que passa sobre a origem do cilindro. (Modificado de (RUS, 2010)).

A Fig. (3 (a)) mostra a representação em 3 dimensões do espaço de cores HSV e destaca os eixos H, S e V como parte de um sistema de coordenadas cilíndricas. A tonalidade equivale à componente angular do sistema e varia de 0 ° a 360 °, a saturação varia entre 0 e 1 e é equivalente à componente radial perpendicular ao eixo vertical que passa na origem e, por fim, a componente do valor varia também entre 0 a 1 e é a componente vertical do sistema. A Fig. (3 (b)) corresponde à área lateral do cilindro abrangendo todo o intervalo das componentes H e V para uma saturação máxima de 1 como é destacado nos lados da figura. Podem ser facilmente identificadas as cores vermelha, verde e azul para os valores 0 °, 120 ° e 240 ° da componente H, respectivamente. Na verdade, para as componentes S e V sendo 1, que equivale ao círculo externo da área do topo do cilindro, encontram-se as cores primárias e secundárias – vermelho, amarelo, verde, ciano, azul e magenta – como também qualquer mistura linear de pares dessas. A Fig. (3 (c)) foi adquirida fazendo um corte transversal no cilindro no plano da componente $V = \frac{1}{2}$ e ajuda a entender um pouco do comportamento do espaço de cores para as variações das componentes V e S, ou seja, pode-se inferir ao se comparar a Fig. (3 (b)) e a Fig. (3 (c)) que a diminuição da componente do valor mostra tons mais escuros para uma mesma

componente S e H e que a diminuição da saturação, mantendo-se as demais componentes iguais, gera uma cor em nível de cinza. Por último, a Fig. (3 (d)) foi obtida através de um corte longitudinal na Fig. (3 (a)), o que gerou uma imagem interessante cujas três componentes do sistema cilíndrico variam e seus efeitos são vistos na coloração da mesma. Pode-se notar claramente a cor primária vermelha e a cor secundária ciano para saturação e valor máximos e para a tonalidade de $0\,^\circ$ e $180\,^\circ$, respectivamente. Também é interessante notar que o eixo da componente V para saturação igual a 0 corresponde à escala de níveis de cinza, onde o valor mínimo $V = 0$ é a cor preta e o valor máximo $V = 1$ representa a cor branca.

O benefício do espaço de cores HSV não se limita a ser um sistema de cores bem mais intuitivo do que o RGB, mas também possui a facilidade na conversão para o espaço RGB: não há funções trigonométricas ou funções que exijam bastante recurso computacional (SMITH, 1978). Este fato possibilita um tempo menor de processamento em nível de *pixel* e também o processamento de imagens maiores ou a uma maior taxa de quadros, quando se trata de uma sequência de imagens.

Para uma cor descrita no espaço RGB e normalizada de forma que as componentes do *pixel* estejam no intervalo entre 0 e 1 para o menor e maior valor, respectivamente, as componentes para o espaço de cores HSV são obtidas através das seguintes relações:

$$H = \begin{cases} 60 \times \frac{G-B}{MAX-MIN}, & se \quad MAX = R \quad e \quad G > B \\ 60 \times \frac{G-B}{MAX-MIN} + 360, & se \quad MAX = R \quad e \quad G < B \\ 60 \times \frac{B-R}{MAX-MIN} + 120, & se \quad MAX = G \\ 60 \times \frac{R-G}{MAX-MIN}, & se \quad MAX = B \end{cases} \tag{2.1}$$

$$S = \frac{MAX - MIN}{MAX} \tag{2.2}$$

$$V = MAX \tag{2.3}$$

Onde MAX e MIN correspondem ao maior e o menor valor dentre os componentes do espaço RGB, respectivamente, e os valores encontrados para o novo espaço HSV estão contido entre 0 e 360 para a componente H, e 0 e 1 para S e V.

2.3 Visão Estéreo

O princípio básico da visão estéreo é a identificação das coordenadas 3D de um determinado ponto a partir do conhecimento das coordenadas desse mesmo ponto em duas

ou mais câmeras previamente calibradas. Isso é feito tendo em vista que as coordenadas 2D desse ponto na imagem, juntamente com a informação da calibração da câmera permitem a determinação de um raio único que parte do ponto especificado até o objeto a que se refere. A intersecção de dois raios únicos fornecidos por duas câmeras fornece a informação necessária para a obtenção da propriedade de profundidade existente no domínio 3D. Sistemas de visão computacional que se beneficiam do conhecimento de sua geometria relativa pra obter a profundidade a partir de duas diferentes visões da mesma cena é um assunto que tem gerado bastante pesquisa atualmente e sua metodologia pode ser resumida em três passos (SONKA; HLAVAC; BOYLE, 2008):

- Calibração da câmera
- Estabelecimento de correspondências entre pares de pontos das imagens direita e esquerda
- Reconstrução 3D

2.3.1 Modelo de Câmera Estenopeica

O modelo de câmera estenopeica, ou no inglês *camera pinhole*, usa a projeção perspectiva como forma de mapeamento geométrico do mundo 3D real para a imagem 2D da câmera como método de aquisição de imagens (Fig. 4).

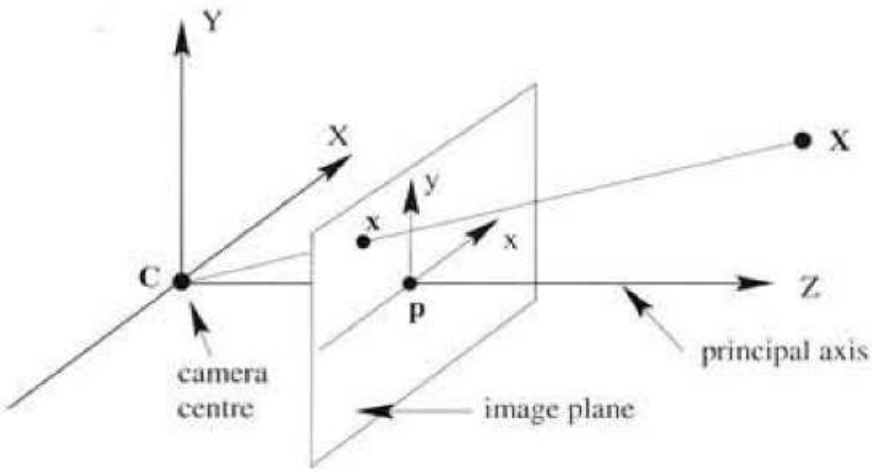


Figura 4 – Modelo de Câmera Estenopeica (SANTOS, 2012).

A Fig. (4) ilustra o modelo de câmera estenopeica. Na figura, o ponto C é o centro de projeção, também comumente chamado de centro da câmera ou centro óptico. A distância focal corresponde ao espaço entre o centro de projeção e o plano da imagem ou plano focal. O eixo que passa pelo ponto C e atravessa perpendicularmente o plano da

imagem é chamado de eixo principal ou eixo óptico. Por fim, o ponto onde o plano focal intercepta o eixo principal é conhecido como ponto principal (SANTOS, 2012).

2.3.2 Geometria Epipolar

A geometria epipolar estabelece a relação geométrica entre duas vistas de uma mesma cena capturadas por duas câmeras diferentes. Essa geometria é usada na busca de pontos coincidentes para algoritmos de visão estéreo visto que as câmeras apresentam imagens de uma mesma cena estática, porém posicionadas em diferentes pontos (SANTOS, 2012).

A Fig. (5 (a)) mostra, a partir de duas imagens diferentes que representam a mesma cena e um ponto X no espaço o qual se deseja analisar, que a correspondência da representação desse ponto nas duas imagens dada como x na imagem esquerda e como x' na imagem da direita é feita com o auxílio de um plano obtido pelos centros ópticos das duas imagens e o ponto no espaço X, conhecido como plano epipolar π.

Detectado o plano epipolar π, pode-se notar que o ponto X, sua representação na imagem da esquerda x e o centro óptico também da imagem esquerda geram um raio que atravessa esses pontos e que a representação de todos os pontos sobre este raio é visto na imagem da direita sobre a linha l'.(Fig. 5 (b)).

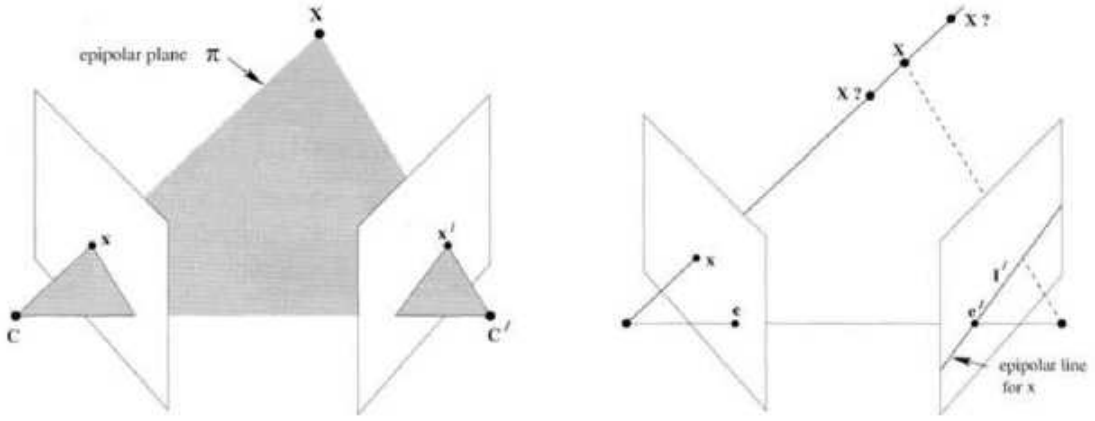


Figura 5 – Correspondêcia entre pontos através da geometria epipolar (SANTOS, 2012).

A partir dos conceitos apresentados sobre a geometria epipolar e as ilustrações referentes à Fig. (5), as entidades geométricas usadas nessa representação são definidas a seguir (SANTOS, 2012):

- **Linha de base**: Linha que passa pelos centros ópticos das duas imagens.
- **Epipolo**: Ponto da linha de base que intersecta o plano da imagem.

- **Plano epipolar** π: É o plano obtido pelos centros de projeção das duas imagens e o ponto em questão no espaço, ou seja, para cada ponto no espaço um plano epipolar diferente é obtido.
- **Linha epipolar**: É a linha obtida da intersecção do plano epipolar com o plano da imagem.

2.3.3 Configuração Retificada de Duas Câmeras

Um arranjo especial da geometria epipolar quando os planos das imagens são coincidentes é a configuração retificada, também conhecida como configuração canônica e câmera retilínea (Fig. 6). Nesse caso, a linha de base, formada pelos centros de projeções das duas imagens, se torna paralela ao plano de imagem e logo não a intersecta, levando os epipolos para o infinito. Como consequência desse alinhamento dos planos das imagens, as linhas epipolares são paralelas às linhas das imagens, o que facilita o cálculo das correspondências. Para tanto, também se assume que os parâmetros intrínsecos de calibração são os mesmos (SONKA; HLAVAC; BOYLE, 2008).

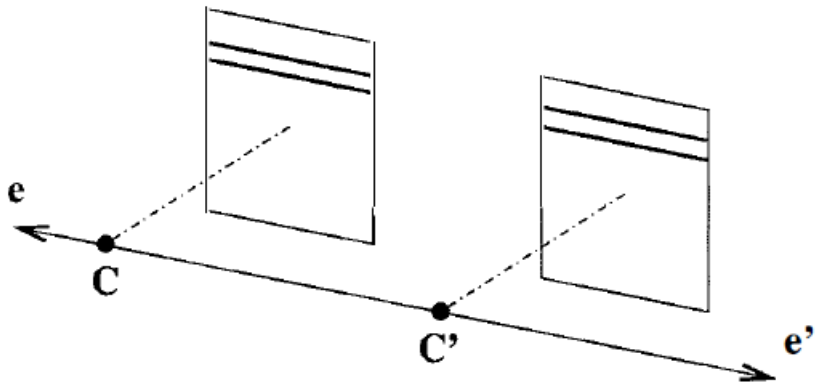


Figura 6 – Configuração rectificada para visão estéreo (SONKA; HLAVAC; BOYLE, 2008).

Considerando a configuração retificada, pode-se encontrar a profundidade de um determinado ponto X no espaço de forma simplificada. A Fig. (7) apresenta uma visão superior do arranjo retificado onde os centros ópticos são separados por uma distância de linha de base de $b = 2h$. A representação do ponto X no espaço é dada pelos eixos locais (u, v) e (u', v') nas imagens da esquerda e da direita, respectivamente. O eixo z na figura representa a distância frontal entre a as câmeras e um determinado ponto no espaço cujo ponto igual a zero corresponde ao plano dos centros ópticos. O eixo x corresponde à distância horizontal e seu ponto igual a zero é a metade da linha de base, por fim a componente y não é mostrada uma vez que ela está perpendicular ao plano da página (SONKA; HLAVAC; BOYLE, 2008).

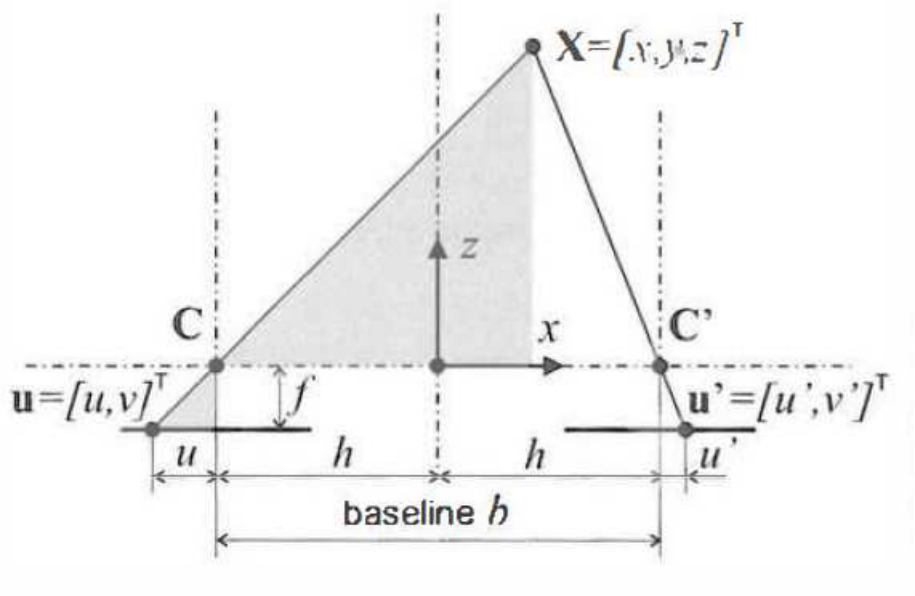


Figura 7 – Geometria da configuração retificada de duas câmeras(SONKA; HLAVAC; BOYLE, 2008).

Observando as componentes u e u', torna-se claro que há uma **disparidade** devido à diferença de posição das duas câmeras. A partir dessa disparidade e, utilizando semelhança de triângulos na representação do arranjo retificado da Fig. (7), é possível determinar a profundidade do objeto X, ou seja, sua componente z no espaço.

Ainda na Fig. (7), nota-se que os segmentos de reta $\overline{uC}$ e $\overline{CX}$ são as hipotenusas dos triângulos retângulos esmaecidos. Assim, usando semelhança de triângulos é possível se obter a seguinte relação:

$$\frac{u}{f} = -\frac{h+x}{z} \tag{2.4}$$

Analogamente, a Eq. (2.5) foi obtida da semelhança de triângulos gerada da projeção do ponto na imagem da direita:

$$\frac{u'}{f} = \frac{h-x}{z} \tag{2.5}$$

Isolando x na Eq. (2.4) e substituindo na Eq. (2.5), pode-se obter a seguinte fórmula para profundidade do ponto X:

$$\begin{aligned} z &= \frac{2hf}{u'-u} \\ &= \frac{bf}{u'-u} \\ &= \frac{bf}{d} \end{aligned} \tag{2.6}$$

Onde b é a distância entre as câmeras, f a distância focal e d a disparidade dos pontos correspondentes nas imagens.

Da Eq. (2.6), pode-se notar que se a disparidade $(u - u') \rightarrow 0$, então $z \rightarrow \infty$, assim uma disparidade 0 significa que a distância frontal das câmeras para o ponto analisado é, para fins práticos, infinita. Assim, a redução de erros em z pode ser obtida através de uma distância da linha de base grande e para pontos do cenário próximos ao arranjo estéreo retificado (SONKA; HLAVAC; BOYLE, 2008).

2.4 Hardware Reconfigurável

Hardware reconfigurável é um meio flexível onde se pode implementar circuitos integrados. Possibilita uma configuração (e reconfiguração, na maioria das plataformas) pós-fabricação de modo que um único hardware possa ser usado para uma grande variedade de implementações(GARCIA et al., 2006).

Dispositivos de *hardware* reconfigurável são constituídos por um conjunto de blocos lógicos e de roteamento que são configurados por uma memória. Basicamente, os valores armazenados nessa memória são os responsáveis por determinar a lógica e as conexões estabelecidas. Assim, nos blocos de roteamento, os valores armazenados nessas memórias de configuração determinam se um fio entre determinado bloco lógico estará conectado ou não e, para os blocos lógicos, a memória de configuração determina as equações lógicas a serem implementadas através de Unidades Lógicas Aritméticas (ULAs) ou de tabelas de busca, LUTs (*Look-up Tables*)(GARCIA et al., 2006).

2.4.1 FPGAs

FPGAs são dispositivos de *hardware* reconfigurável. A sua estrutura básica e componentes essencias podem ser vistos na Fig. (8). Pode-se observar a presença de três blocos principais. Os blocos lógicos são formados por um conjunto de blocos de granularidade-fina (*fine-granularity*), que executam operações com um 1 *bit* de largura e implementam a lógica da aplicação. Tais blocos lógicos são dispostos de forma matricial e são interconectados através dos blocos de conexão e chaveamento possibilitando que os blocos elementares de lógica se conectem apropriadamente para implementar a lógica global desejada. Nas margens da Fig. (8) pode-se observar os blocos de entrada e saída (I/O), que possibilitam a interface entre os componentes internos do FPGA e dispositivos externos (BAIELY, 2011).

Por fim, a Fig. (8), mostra dois blocos de controle existentes em FPGAs, o primeiro é responsável por receber a lógica do usuário, que será configurada nos blocos lógicos, e o segundo é um bloco de gerenciamento de *clock*, que sincroniza e disponibiliza o sinal do relógio para as demais partes do circuito.

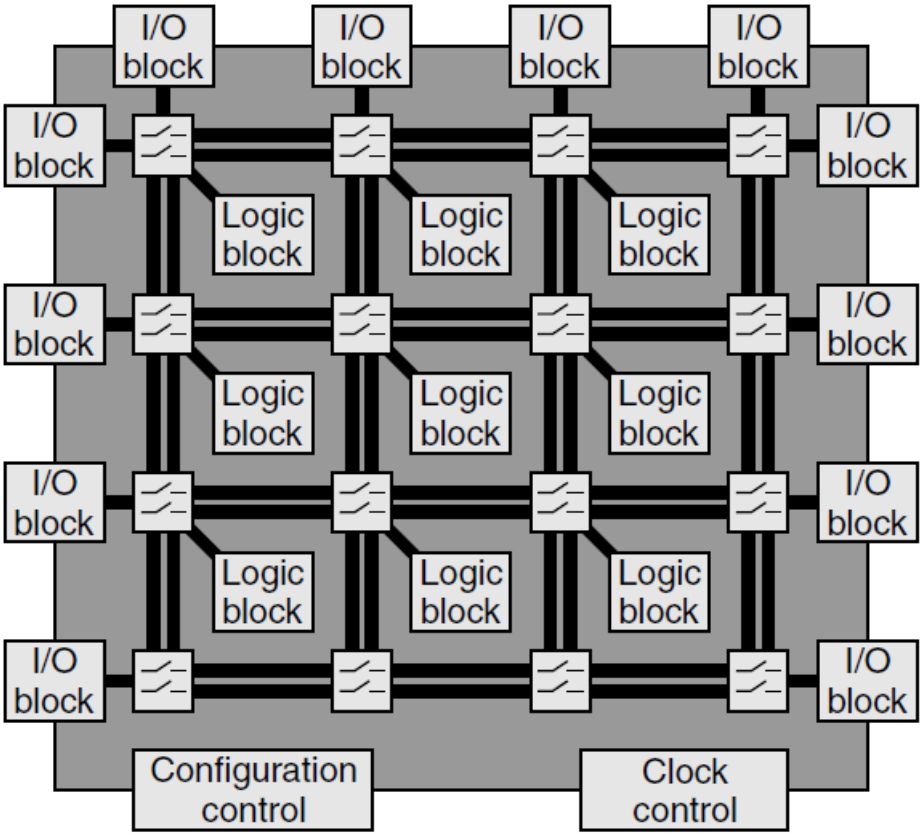


Figura 8 – Arquitetura Básica de um FPGA (BAIELY, 2011).

A menor unidade lógica encontrada em um FPGA é a célula lógica, bloco de granularidade fina que compõem os blocos lógicos. É composta basicamente por uma LUT juntamente com um *flip-flop* (nas arquiteturas modernas são encontradas LUTs de 4, 5 ou 6 entradas, dois *flip-flops* e muxes) para armezenar o bit de saída (BAIELY, 2011). A Fig. (9 (b)) mostra a implementação de uma LUT de três entradas através de um multiplexador. Tal LUT implementa a função representada pela tabela verdade da Fig. (9 (a)). Cada entrada do multiplexidor equivale ao valor dessa função para uma determinada combinação das variáveis da função, inseridas na LUT através dos *bits* de seleção.

Todos os elementos lógicos e de roteamento em um FPGA são programados via memória de configuração, podendo ser implementada através de memórias *Flash*, SRAM ou por anti-fusíveis (HAUCK; DEHON, 2008). Na Fig. (9 (b)), tais pontos que configuram a LUT mostrada equivalem às entradas dos multiplexadores e são obtidos através da tecnologia SRAM.

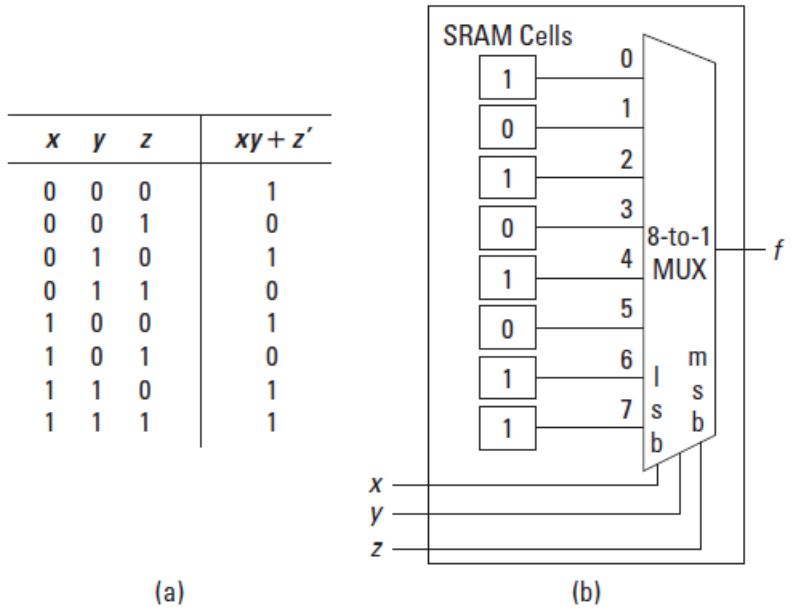


Figura 9 – Representação de uma função booleana em um FPGA. (a) Tabela verdade e (b) Implementação através de uma LUT (SASS; SCHMIDT, 2010).

2.4.2 FPGAs vs. ASICs

A flexibilidade advinda da lógica programável dos FPGAs tem seu custo. Uma comparação entre circuitos implementados em FPGAs e circuitos integrados de aplicação específica (ASICs) mostra que os FPGAs sempre usarão mais silício, será mais lentos e vão dissipar mais potência (BAIELY, 2011).

Um ASIC sempre terá uma área menor por uma série de motivos. Primeiro, um circuito dedicado será formado apenas pelos componentes necessários para implementar a sua lógica enquanto um circuito programável será formado por partes que não serão usadas, ou que não serão usadas da melhor forma possível. Ainda, a interconexão de circuitos com lógica configurável é feita através de blocos de interconexão a fim de possibilitar ou não a conexão de blocos lógicos enquanto os circuitos dedicados dispensam esse tipo de conexão, sendo usados apenas fios que conectam os nós desejados para determinada lógica. Outro ponto refere-se ao bloco de lógica em si, para a lógica reconfigurável este bloco é inevitavelmente maior do que o bloco para uma lógica específica (BAIELY, 2011).

Em relação à velocidade, uma implementação em FPGA sempre será mais lenta que uma feita em ASIC. Em um ASIC é possível se colocar os componentes o mais próximos possível a fim de se minimizar os *delays*, já nos FPGAs os blocos de lógica ficarão mais afastados devido aos blocos de conexão, aumentando esses *delays*. Além do mais, devido à maior área do circuito programável, este terá mais capacitância e, portanto, um menor *clock* máximo associado (BAIELY, 2011).

O aumento na potência em implementações em circuitos programáveis é devido ao aumento de capacitância e da maior quantidade de transistores a serem chaveados (BAIELY, 2011). A Eq. (2.7) mostra como é obtida a dissipação de potência dinâmica em um circuito:

$$P_{dyn} = \alpha C_L V_{DD}^2 f_{clk} \tag{2.7}$$

Onde α é o fator de atividade de chaveamento e C_L é a capacitância de carga.

Apesar do que foi apresentado nos parágrafos anteriores, FPGAs possuem vantagens se comparado aos ASICs. As máscaras usadas na fabricação dos circuitos dedicados e os custos de projeto são bem maiores comparadas com os custos de um circuito em plataforma reconfigurável para uma quantidade menor de unidades, o que torna os ASICs economicamente mais vantajosos apenas para grandes quantidades (Fig. 10). Os FPGAs também oferecem um *time to market* melhor pois possuem um tempo de projeto menor e oferecem a possibilidade de modificação em etapas avançadas do ciclo de projeto. Devido ainda à propriedade de ser programável em campo, produtos baseados em FPGA possuem maior tempo de vida (BAIELY, 2011).

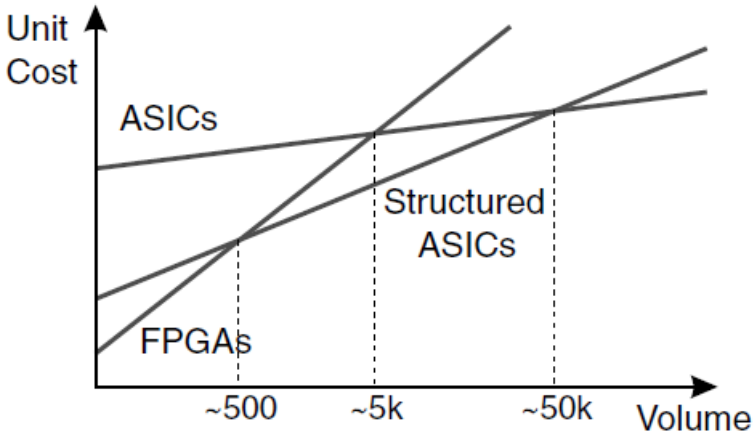


Figura 10 – Custos relativos entre FPGAs, ASICs e ASICs estruturados (BAIELY, 2011).

ASICs estruturados (Fig. 10) é um híbrido entre as duas tecnologias discutidas, a fim de se obter propriedades mescladas. Consiste de um conjunto pré-definido de transistores com conexões adicionadas posteriormente (BAIELY, 2011).

2.4.3 Vantagens do Uso de Hardware Reconfigurável

O projeto de hardware é marcado por um conflito entre desempenho do sistema, custo, consumo de potência, e outros parâmetros relacionados à implementação (LENART, 2008). Um modo de lidar com isso é o uso de múltiplas unidades de processamento

paralelos a fim de gerar um ganho de desempenho sem comprometer ou minimizando a interferência com as demais variáveis de projeto. Para obter uma aceleração significativa de uma implementação em *harware* reconfigurável sobre uma em *software*, é necessário que uma parte significante do algoritmo possa ser implementado em paralelo (BAIELY, 2011).

Todos os processadores de *software* comerciais usam o modelo de buscar-executar um conjunto de instruções contidas em uma memória endereçável (SASS; SCHMIDT, 2010). Dessa forma, as operações são executadas sequencialmente na forma de instruções, processadas de forma ordenada no processador de propósito geral, simplesmente ignorando quaisquer paralelismos que por ventura existam na implementação. Por outro lado, as tarefas em dispositivos configuráveis são implementadas espacialmente, através de operações primitivas (DEHON, 2000). Este modelo é conhecido como fluxo de dados ou *data-flow*.

Uma forma de exemplificar as diferenças entre os modelos sequencial e de fluxo de dados referentes aos processadores genéricos de *software* e as arquiteturas reconfiguráveis, respectivamente, pode ser vista através de suas respectivas implementações para a primitiva descrita na Eq. (2.8):

$$R0 + R1 + R2 \rightarrow Acc \tag{2.8}$$

Onde $R0$, $R1$ e $R2$ se referem a registradores que armazenam operandos de uma adição e Acc é o registrador especial que guardará o resultado dessa soma, para uma representação de circuitos em *Register-Transfer Level* (RTL).

A Fig. (11) mostra o modelo de computação clássico, referido como modelo computacional de *von Neumann*, onde as operações são executadas sequencialmente e controladas por um controlador que busca e decodifica as instruções em uma memória de programa e então multiplexa as entradas da ULA para o processamento das instruções. A Fig. (12) mostra um exemplo da mesma implementação usando o conceito de fluxo de dados, ou modelo não-sequencial, onde a operação é "espalhada" espacialmente e o conceito de sequência de instruções já não é aplicado.

A consequência mais significativa dessa abordagem diferenciada nos dispositivos reconfiguráveis é um ganho significativo de desempenho em comparação com os processadores de *software* devido à enorme redução de ciclos de relógio para acesso das memórias (KILTS, 2007).

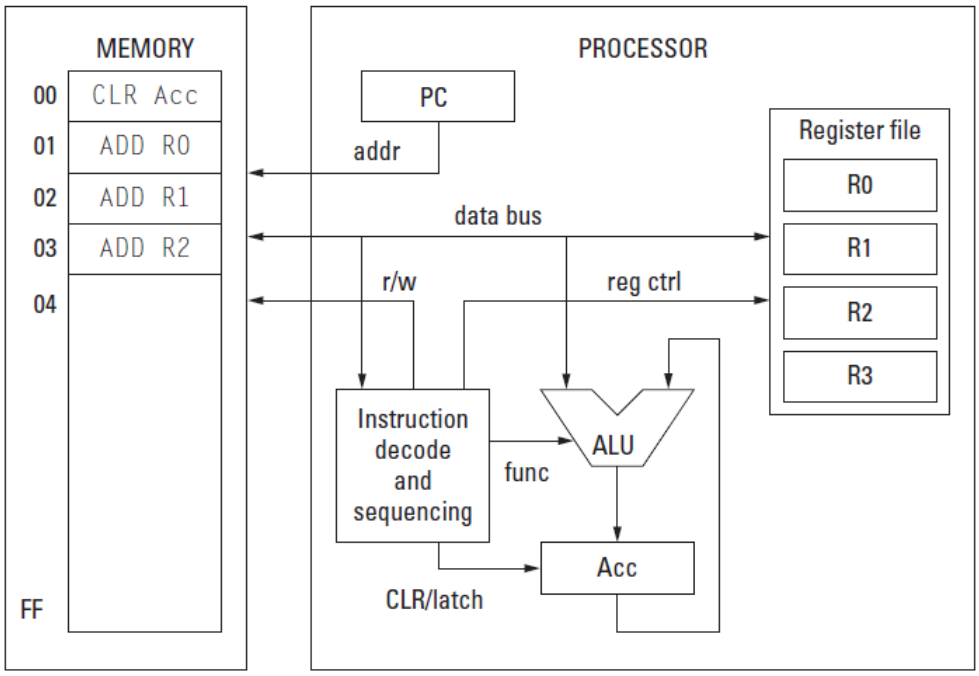


Figura 11 – Exemplo de uma implementação do modelo computacional sequencial (SASS; SCHMIDT, 2010).

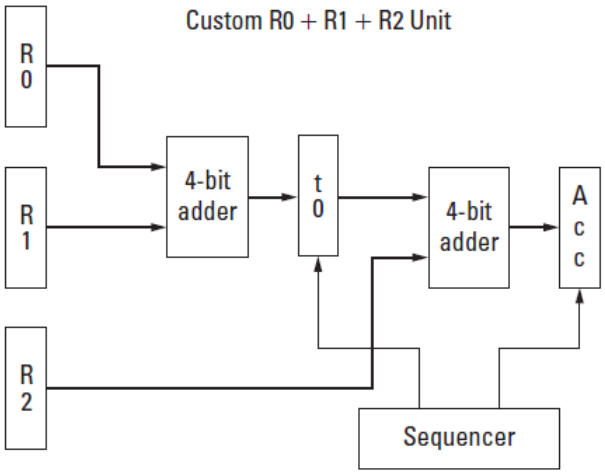


Figura 12 – Exemplo de uma implementação do modelo computacional não-sequencial ou de fluxo de dados (SASS; SCHMIDT, 2010).

3 Implementações

3.1 Plataformas de Hardware e Software

3.1.1 Placa Educacional e de Desenvolvimento DE2 e DE2-70

O trabalho de segmentação por cores (BITTENCOURT, 2012) que é utilizado neste trabalho foi desenvolvido e implementado na plataforma FPGA DE2 da fabricante *Terasic Technologies*, empresa especializada em projetos de placas de desenvolvimento baseadas em FPGA. O kit de desenvolvimento Altera DE2 possui um FPGA Cyclone II EP2C35 da Altera e suas informações principais são apresentadas na Tab. (1).

Tabela 1 – Recursos disponíveis na placa de desenvolvimento DE2 (ALTERA, 2012).

Recurso	Descrição
FPGA	• FPGA Cyclone II EP2C35F672C6 da Altera com dispositivo de configuração serial EPCS16 de $16Mbit$
Interfaces de I/O	• USB *Blaster* para configuração do FPGA • CODEC para entrada e saída de áudio de $24-bits$ entrada para microfone • Saída de vídeo VGA com DAC de $10-bits$ • Entrada de vídeo nos formatos NTSC, PAL e Multi-formato • Comunicação serial RS232 • Porta para comunicação por infravermelho • Conector para *mouse*/teclado PS/2 • Controlador *Ethernet* $10/100Mbits/s$ • Controlador USB *Hosta/Slave* com conectores USB tipo A e tipo B • Duas portas de expansão de 40 pinos cada
Memória	• Memória SRAM de $512-Kbytes$ • Memória SDRAM de $8-Mbytes$ • Memória *Flash* de $4-Mbytes$ • Entrada para cartão SD
Displays	• 8 *displays* de 7-segmentos • Display LCD de 2 linhas por 16 colunas
Chaves e LEDs	• 18 chaves • 18 LEDs vermelhos • 9 LEDs verdes • 4 chaves de *push-button*
Clocks	• Osciladores de 50 MHz e 27 MHz

O presente trabalho propõe implementações que vão além do sistema de segmentação por cores que usa uma câmera para aquisição de imagens e um *display* LCD (*Liquid-*

Crystal Display) para apresentação da imagem segmentada. Para implementar o sistema de visão estéreo foi necessário o uso de uma câmera adicional e, consequentemente, a duplicação de todos os módulos usados na aquisição das imagens em tempo real. Por outro lado, o *display* não foi mais necessário, uma vez que o resultado não era mais uma imagem segmentada e sim a distância frontal para guiar um robô móvel.

Da caracterização da arquitetura proposta em (BITTENCOURT, 2012) pôde ser visto que o consumo de *bits* de memória do sistema é de 75% com apenas o processamento da imagem de uma câmera, logo a duplicação da arquitetura em questão extrapolaria por muito a quantidade de memória disponível caso fosse implementado na placa de desenvolvimento DE2.

A solução encontrada para essa limitação de memória foi a troca da plataforma usada para uma plataforma similar, porém com maior recurso de armazenamento. Para isso, foi usada a plataforma FPGA DE-70, que é uma versão modificada da placa DE2 com mais memória e um FPGA maior. As diferenças entre as duas placas de desenvolvimento são, literalmente, uma maior quantidade de memória dado por duas memórias SDRAM separadas de 32 MB cada, 2 MB de SSRAM e uma memória *Flash* de 8 MB e juntamente com um ganho no poder computacional devido ao FPGA Cyclone II EP2C70, se comparando com os recursos mostrados na Tab. (1).

Os demais periféricos da placa de desenvolvimento DE2 se mantêm para a DE2-70. Uma fotografia do kit DE2-70 e a indicação dos seus componentes pode ser vista na Fig. (13).

3.1.2 Kit de Desenvolvimento da Câmera Digital e *Display* LCD *Touch Screen*

Para a aquisição das imagens do sistema estereoscópio foram usados dois kits de câmera digital de resolução de 5 *Megapixel* TRDB_D5M da *Terasic Technologies* (Fig. 14).

Apesar de não ser utilizado no sistema final de visão estéreo, foi feito o uso de um *display* LCD a fim de verificar a correta segmentação por cores usada como base da detecção do objeto de interesse na fase de validação e testes do sistema. Para tanto, foi usada a placa com módulo do *display* LCD *touch screen* TRDB_LTM de 4,3 polegadas também da *Terasic Technologies* (Fig. 15).

Os modelos das câmeras e do modulo do *display touch screen* usados são compatíveis e comumente usados com diversos kits de desenvolvimento baseados em FPGAs da Altera como é o caso de ambas as plataformas DE2 e DE2-70 utilizadas nesse trabalho. O uso desses componentes não é só suportado na plataforma utilizada, mas também indicado, uma vez que ambos os *datasheets* disponibilizam demonstrações de arquiteturas de *hardware* e *software* específicas para os kits DE2 e DE2-70 de sistemas multimídia

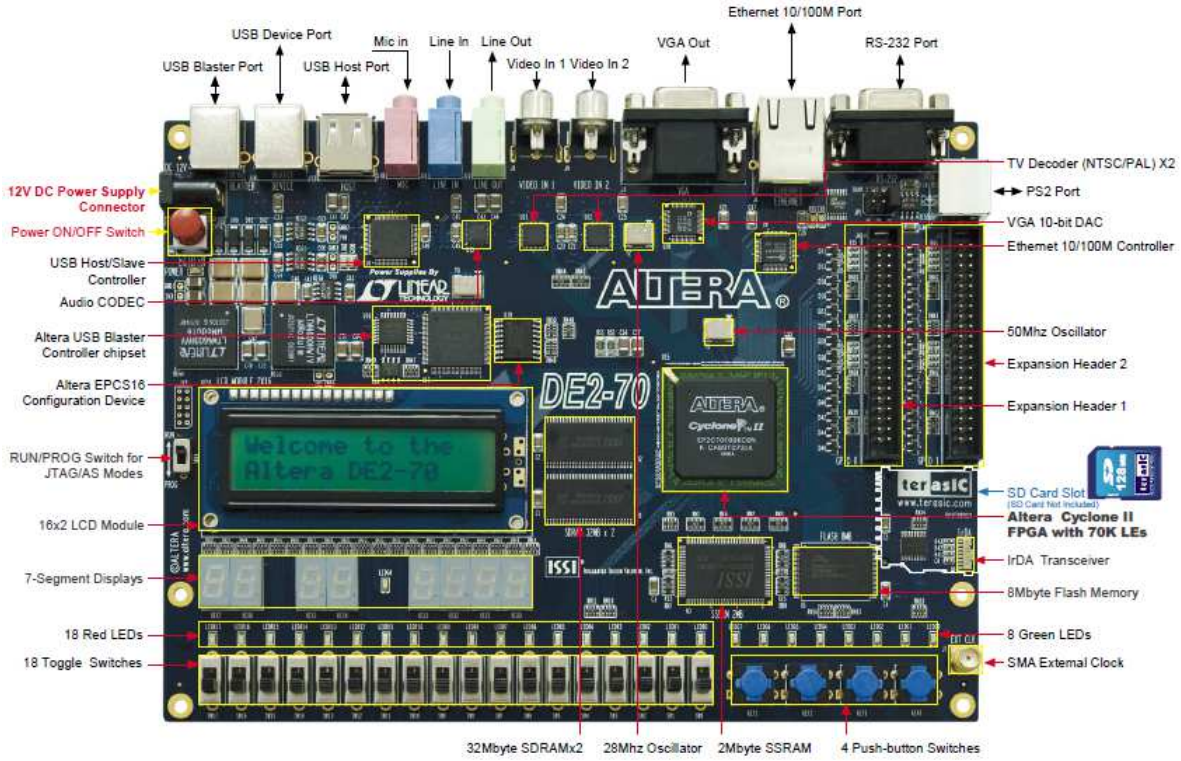


Figura 13 – Placa DE2-70 (TERASIC, 2009).

envolvendo o uso da câmera digital e do *display* LCD, o que possibilita um ponto de partida para implementações usando as plataformas apresentadas. É importante salientar que a arquitetura de segmentação por cores apresentada em (BITTENCOURT, 2012) está baseada nos módulos de aquisição de imagens disponibilizados como demonstrações dos kits de desenvolvimento da câmera digital e do módulo do *display* LCD.

3.1.3 Software de Desenvolvimento Quartus II

No presente trabalho foi utilizada a versão 13.0 da ferramenta de desenvolvimento Quartus II da Altera. É possível implementar uma solução utilizando várias linguagens de descrição de *hardware* diferentes simultaneamente no mesmo projeto como o *Verilog*, o VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) e o AHDL (*Altera Hardware Description Language*). As duas primeiras são as linguagens de descrição de *hardware* mais empregadas e a última é uma HDL própria desenvolvida pela Altera. A ferramenta Quartus permite a simulação comportamental do projeto implementado como também é capaz de desempenhar todo o fluxo de projeto para FPGAs fazendo a síntese lógica da arquitetura proposta, desempenhando o posicionamento e mapeamento da mesma, gerando o *stream* de bits e, por fim, programando o FPGA via comunicação USB.

A Fig. (16) mostra um projeto sendo implementado no ambiente de desenvolvi-

Figura 14 – Kit de desenvolvimento de câmera digital TRDB_D5M da *Terasic Technologies* (TERASIC, 2008).

Figura 15 – Placa com módulo do *display* LCD *touch screen* TRDB_ LTM da *Terasic Technologies* (TERASIC, 2003).

mento Quartus II da Altera. No bloco central da imagem pode-se ver um arquivo de descrição em *Verilog* em modo de edição, no campo superior esquerdo são mostradas as entidades do projeto seguindo a hierarquia da arquitetura e logo abaixo notam-se as etapas do fluxo de projeto, as quais são desempenhadas pelo *software* após a descrição do projeto.

O sistema foi desenvolvido usando um computador portátil equipado de um pro-

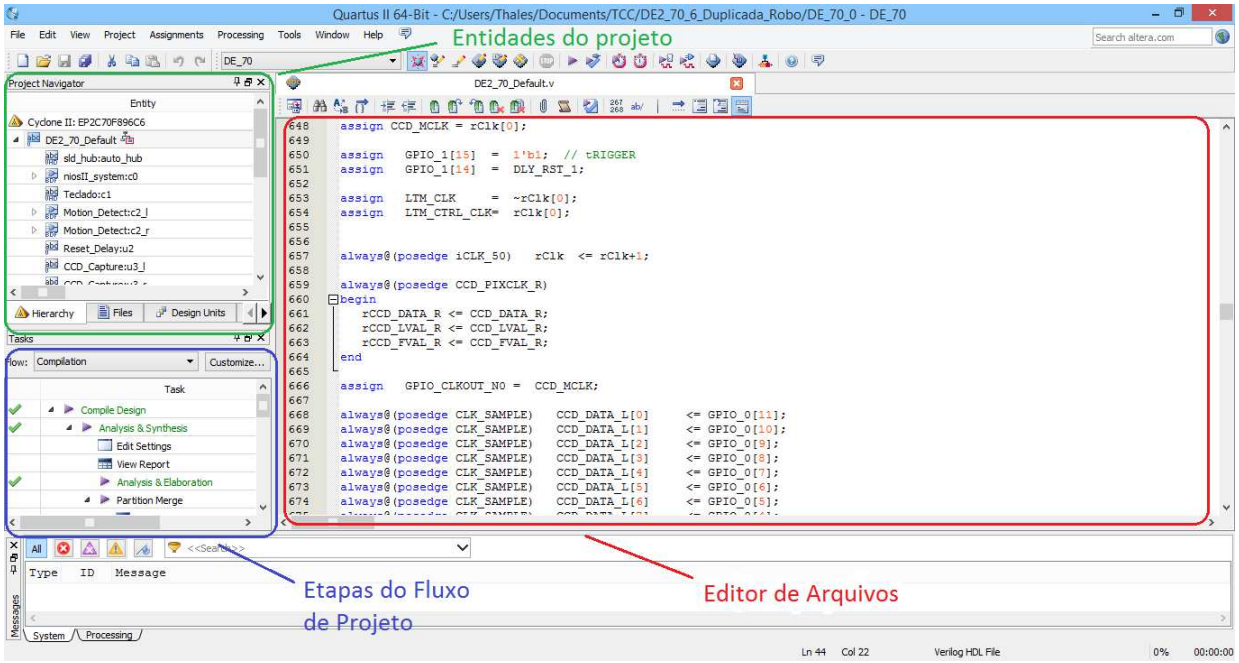


Figura 16 – Exemplo de projeto no ambiente de desenvolvimento Quartus II da Altera.

cessador *intel i7* com quatro núcleos de processamento, provido de $6MB$ de memória *cache* e operando a uma frequência máxima de $3.4GHz$. O sistema computacional possui ainda $16GB$ de memória RAM e $250GB$ de memória *Solid-State Drive* (SSD). O sistema operacional rodando na máquina é o *Windows* 8.0.

3.1.4 Qsys

Os FPGAs de ambos os kits de desenvolvimento DE2 e DE2-70 permitem o uso de um processador de *software* embarcado a ser usado em conjunto com a arquitetura de *hardware* que está sendo implementada. O processador NIOS II é classificado como *soft core* porque não está fisicamente construído nos FPGAs em questão, mas pode ser implementado usando os elementos lógicos programáveis e assim ser acoplado ao projeto.

O processador NIOS II também possui a qualidade de ser configurável de acordo com a necessidade da aplicação. Para isso pode-se usar o *software* Qsys para construir as características de *hardware* do processador, adicionando componentes e configurando como estes estarão conectados (Fig. 17). Como pode ser visto na imagem é possível especificar todos os periféricos e recursos do sistema, tais como pinos de I/O de tamanho variáveis, memórias *on-chip* ou blocos de SDRAM, módulos do *timer*, relógio do sistema, entre outros.

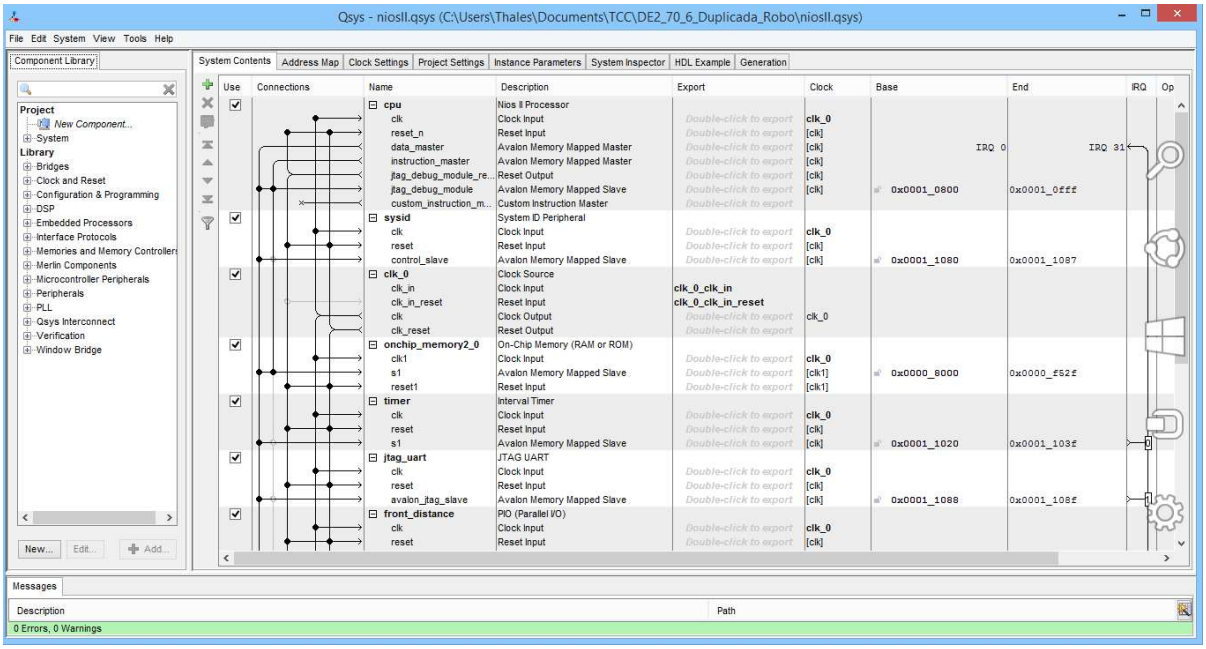


Figura 17 – Exemplo de construção da arquitetura do processador NIOS II no ambiente Qsys

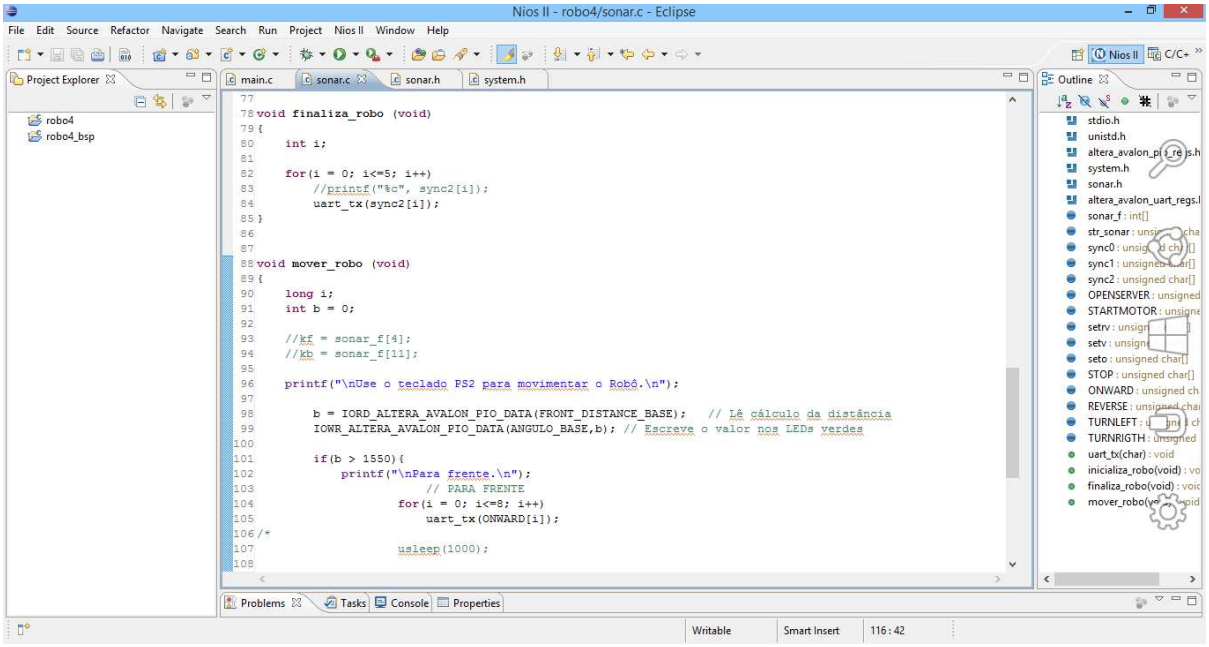
3.1.5 NIOS II Software Build Tools for Eclipse

Uma vez gerada a arquitetura do processador NIOS II que atenda os requisitos do projeto, é necessário instanciar o sistema produzido no módulo principal do *Quartus II*, assim o processador de *software* construído aparece como mais um módulo integrante do sistema.

Porém para realmente ser útil e executar a função desejada, é necessário que se desenvolva uma aplicação para ser executada no processador. Com o auxílio da ferramenta de desenvolvimento NIOS II SBT é possível programar o NIOS II como seria feito para qualquer outra plataforma. Pode-se usar a linguagem de programação C/C++, compilar o código, fazer depuração e baixar o executável para o kit de desenvolvimento usado. A Fig. (18) mostra detalhes do ambiente de desenvolvimento NIOS II SBT. Nota-se no lado esquerdo da interface gráfica a hierarquia dos arquivos que compõem a aplicação, no centro da tela está o editor de texto, no canto direito há um resumo do arquivo em edição como os arquivos cabeçalho que o compõem e as variáveis declaradas, por fim na parte de baixo há um console que mostra mensagens vindas do NIOS e é de grande importância na depuração e no entendimento do código.

3.1.6 Tera Term

Uma das vantagens da se utilizar o processador NIOS II é a praticidade e flexibilidade para se desempenhar certas funções. Um exemplo claro pode ser mostrado na comunicação serial, o NIOS II pode ser programado facilmente, usando os *drivers* gerados




```
77
78 void finaliza_robo (void)
79 {
80     int i;
81
82     for(i = 0; i<=5; i++)
83         //printf("%c", sync2[i]);
84         uart_tx(sync2[i]);
85 }
86
87
88 void mover_robo (void)
89 {
90     long i;
91     int b = 0;
92
93     //kf = sonar_f[4];
94     //kb = sonar_f[11];
95
96     printf("\nUse o teclado PS2 para movimentar o Robô.\n");
97
98         b = IORD_ALTERA_AVALON_PIO_DATA(FRONT_DISTANCE_BASE);   // Lê cálculo da distância
99         IOWR_ALTERA_AVALON_PIO_DATA(ANGULO_BASE,b); // Escreve o valor nos LEDs verdes
100
101         if(b > 1550){
102             printf("\nPara frente.\n");
103                     // PARA FRENTE
104                 for(i = 0; i<=8; i++)
105                     uart_tx(ONWARD[i]);
106 /*
107                 usleep(1000);
108
```

Figura 18 – Exemplo de desenvolvimento de aplicação no ambiente NIOS II SBT

pela ferramenta NIOS II SBT para enviar e receber dados via porta RS-232, existente nos kits de desenvolvimento usados nesse trabalho. Um ponto em que essa funcionalidade gera benefícios no projeto é após o cálculo da distância frontal. Uma vez que nessa etapa do projeto o *display* LCD TRDB_LTM não é mais utilizado, a verificação do correto funcionamento da arquitetura torna-se mais complicada. Assim, medidas relevantes para o projeto podem ser visualizadas em um computador com o *Tera Term*, um simples programa emulador de terminal. Seu uso é bastante simples, para comunicação serial basta escolher qual porta de comunicação do sistema vai ser usada e definir parâmetros básicos do envio ou recebimento de dados, como *baud rate* (taxa de transmissão de dados), tamanho do dado, se há ou não *bit* de paridade e de parada, entre outros. Outra funcionalidade útil é a possibilidade de salvar em um arquivo os dados recebidos e também carregar arquivos existentes na tela do programa. Na Fig. (19) pode se ver a interface simplificada do *Tera Term* com o terminal recebendo dados e acima a barra de menu para configuração e controle do programa.

3.1.7 Plataforma robótica *Pioneer 3-AT*

Uma vez finalizado e validado o sistema de visão estéreo proposto, tal implementação foi aplicada a uma plataforma robótica móvel para rastreamento de um objeto colorido em um ambiente real. Para isso será utilizado o robô *Pioneer* $3-AT$ da *Adept Mobilerobots*, empresa especializada em sistemas robóticos para pesquisa e desenvolvimento (Fig. 20). O robô *Pioneer 3-AT* é provido de quatro rodas impulsionadas por quatro motores e é ideal para aplicações em diversos tipos de terrenos. Possui um microcontrolador com

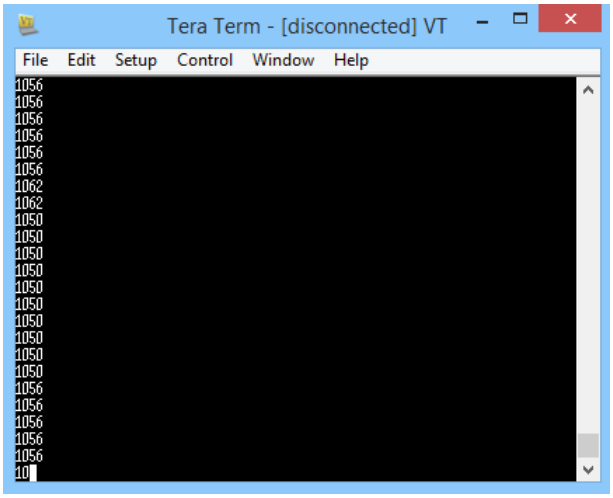


Figura 19 – Exemplo de recebimento de dados usando o emulador de terminal Tera Term

firmware ARCOS embarcado e disponibiliza o pacote de desenvolvimento de *software* para robôs móveis *Pioneer* SDK.

Figura 20 – Plataforma robótica *Pioneer 3-AT* (ADEPT, 2011).

A plataforma robótica tem o sistema de locomoção conhecido como *Skid Steering*, onde os 4 motores do robô estarão ligados ao mesmo tempo. Não há função de escorregar encontrada nos modelos convencionais dos automóveis onde as rodas traseiras são capazes de impulsionar e as dianteiras apenas deslizam, ou seja, não possuem impulsão. Dessa forma, o controlador do motor do ARCOS possibilita diversos tipos de comandos de movimento tais como movimentos de roda independentes (define-se velocidades diferentes para as rodas) e movimentos de translação e rotação usuais (ADEPT, 2010).

O robô *Pioneer 3-AT* possui um perfil trapezoidal de velocidade (Fig. 21). Assim,

quando o robô recebe um comando para se movimentar, ele acelera ou desacelera de acordo com as acelerações máximas definidas (caso o usuário não as tenha definido, a plataforma robótica utiliza constantes armazenadas na memória FLASH) até que os valores máximos de velocidade sejam alcançados.

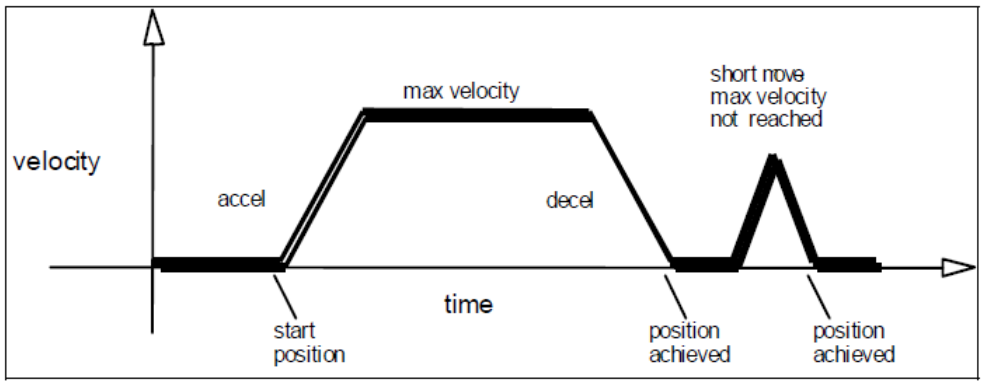


Figura 21 – Perfil de velocidade do robô *Pioneer 3-AT*.

3.2 Arquitetura de Detecção de Cores

3.2.1 Segmentação por Cores em Plataforma Reconfigurável

Este trabalho foi inspirado na solução de segmentação por cores proposta em (BITTENCOURT, 2012). Nesse trabalho prévio, as componentes RGB do *stream* de imagens adquiridas pela câmera são convertidas para o espaço de cores HSV e, então, os *pixels* nessa nova representação de cores passam por uma limiarização para se detectar a cor desejada. A Fig. (22) mostra o resultado obtido em (BITTENCOURT, 2012), onde a Fig. (22 (a)) mostra a cena a qual a câmera estava apontada com o respectivo objeto colorido de interesse e a Fig. (22 (b)) ilustra o resultado da segmentação da Fig. (22 (a)) no *display* LCD TRDB_LTM da *Terasic*. Na Fig. (22 (b)), os *pixels* em azul são os que foram identificados pelo sistema como do objeto procurado e o quadrado vermelho representa o centro geométrico do mesmo.

A Fig. (23) mostra a arquitetura responsável pelo processamento descrito. A colaboração dada por (BITTENCOURT, 2012) se reflete apenas no módulo de segmentação destacado em vermelho na imagem, uma vez que o seu projeto também foi proposto a partir de um trabalho anterior (FERREIRA, 2012). Este projeto usou a mesma plataforma FPGA DE2, o kit de desenvolvimento da câmera TRDB_D5M e o *display* LCD TRDB_LTM e, basicamente, tinha o objetivo de detectar objetos em movimento, calcular o centro geométrico do objeto identificado e apresentar o resultado no *display*. (BITTENCOURT, 2012) mudou a forma de identificação do objeto de interesse, que era detectado através da aplicação do algoritmo de subtração de fundos para a identificação por seg-

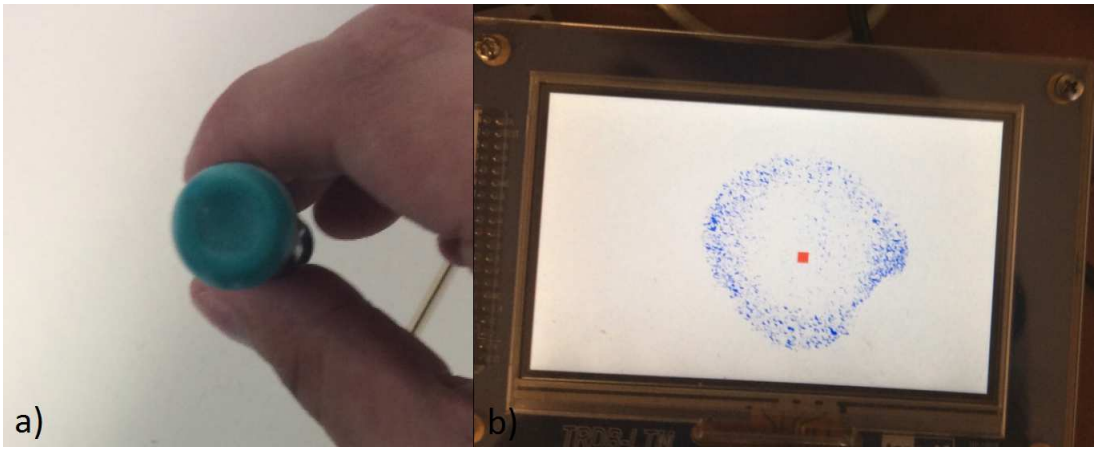


Figura 22 – Resultados da segmentação por cores obtidos em (BITTENCOURT, 2012).

mentação por cores, deixando as demais componentes do sistema, inclusive o módulo de cálculo do centro geométrico, como é mostrado na Fig. (23), sendo a partir desse ponto que o presente trabalho foi proposto.

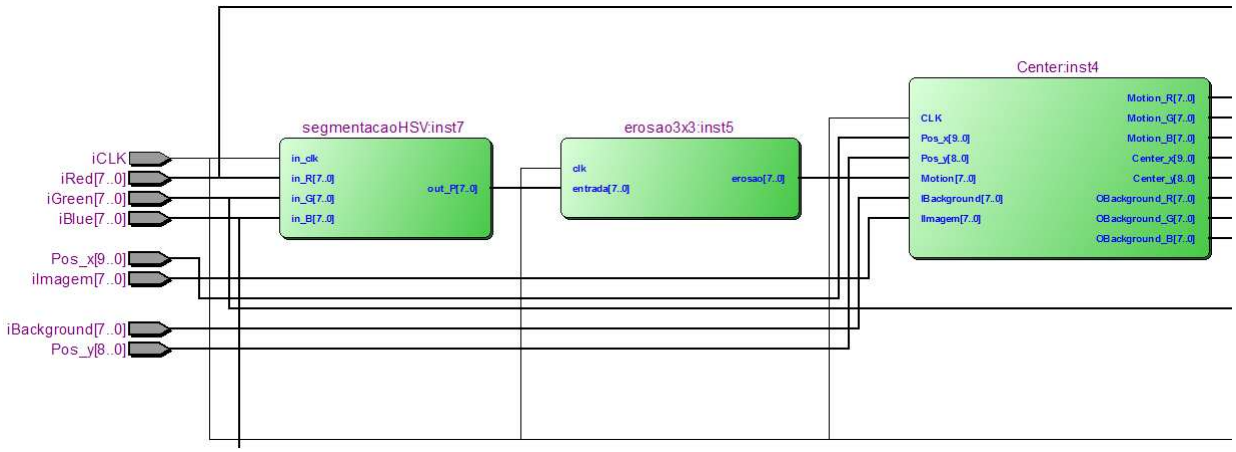


Figura 23 – Módulos de segmentação por cores e cálculo do centro geométrico (BITTENCOURT, 2012).

É importante ressaltar que o código em linguagem de descrição de *hardware* recebido do sistema da Fig. (23) era uma versão não-funcional da segmentação por cores descrita nos parágrafos anteriores. Uma vez que tal implementação estava sendo reusada pela terceira vez (houve um projeto anterior ao (FERREIRA, 2012) cujos códigos de aquisição e visualização foram aproveitados), esta continha arquivos das linguagens *Verilog*, VHDL e módulos descritos por meio de esquemáticos além de uma grande quantidade de código herdado das versões anteriores. Alguns módulos não tinham mais funcionalidade no projeto, tornando difícil o seu entendimento e dificultando o seu aproveitamento. A solução encontrada se deu a partir de uma versão funcional cedida por (FERREIRA, 2012) onde o módulo de segmentação de imagens por cores foi instanciado fazendo as alterações necessárias para torná-lo funcional. Dentre essas alterações podem se destacar a alteração

da fórmula matemática para a mudança do espaço de cores usada em (BITTENCOURT, 2012) para a Eq. (2.1) e a mudança dos valores de limiarização utilizados para detecção das cores desejadas.

3.2.2 Segmentação por Cor Inserida pelo Usuário

A fim de se tornar o sistema da Fig. (23) mais flexível e robusto, foram implementadas alterações no bloco de segmentação a fim de agregar a funcionalidade de escolha da cor a ser segmentada pelo usuário do sistema através do uso das chaves existentes na plataforma FPGA DE2.

Partindo das propriedades do espaço de cores HSV apresentadas na seção 2.2.2 do capítulo anterior e melhor observadas na Fig. (3), foram determinadas seis possibilidades de cores cujos objetos podem ser segmentados pelo sistema. De acordo com a Equação (2.1), a componente H tem o intervalo de $0\,^\circ$ a $360\,^\circ$ e equivale à tonalidade da cor em questão, ou seja, a tonalidade da cor é desacoplada das demais componentes no espaço HSV. Desse modo, notou-se que as cores primárias e secundárias dadas pelo vermelho, verde, azul, ciano, magenta e amarelo tinham espaçamento constante de $60\,^\circ$ na componente de tonalidade e, portanto, geravam uma margem relativamente grande para minimizar o erro na identificação dessas cores, baseando-se exclusivamente na componente H. Esse intervalo entre as cores são importantes quando consideradas as variações de iluminação do ambiente e pequenas variações entre a cor de referência e a cor real do objeto de interesse. A Tabela (2) apresenta as componentes HSV para as cores primárias e secundárias.

Tabela 2 – Componentes HSV para as cores primárias e secundárias.

	H(°)	**S**	**V**
Vermelho	0	1	1
Amarelo	60	1	1
Verde	120	1	1
Ciano	180	1	1
Azul	240	1	1
Magenta	300	1	1

Como já falado anteriormente nesse documento, o trabalho proposto realizou os objetivos descritos em um ambiente acadêmico, validando o sistema final em um ambiente controlado. Tendo isso em vista, pôde-se fazer uso das propriedades do fundo da imagem para dar maior robustez à implementação. Como pode ser visto na Fig. (3), a cor branca ocupa o centro da área da base superior do cilindro referente ao espaço de cores HSV. Assim, pontos brancos em uma imagem real serão aqueles que possuírem a componente V próxima ao valor máximo e a componente S próxima ao valor mínimo, de acordo com a Fig. (3). Enquanto componente H de tonalidade, na prática, é irrelevante podendo assumir qualquer valor desde que as demais componentes estejam nos intervalos destacados.

Representando isso na forma de equação, pode-se gerar a Eq. (3.1) abaixo onde "X" faz referência ao valor lógico *don't care* ilustrando a independência da cor branca em relação à componente de tonalidade do espaço HSV:

$$(H, S, V) = \{X, 0, 1\} \tag{3.1}$$

Isso significa que um sistema que use a cor branca como *background* e deseja segmentar as cores destacadas na Tab. (2) apenas pelos valores de H, eventualmente irá gerar erros e identificará *pixels* no plano de fundo da imagem como coloridos. Porém, o conhecimento dessa situação possibilita a fácil solução do problema simplesmente acrescentando nas condições de detecção desejada não apenas a tonalidade buscada, como também valores altos do componente da saturação.

No nível de sistema, a única modificação feita nos módulos ilustrados na Fig. (23) é a inserção de um *array* de entrada adicional no bloco de segmentação denominado *Sel_Color[2..0]* proveniente das chaves *SW0*, *SW1* e *SW3* do kit de desenvolvimento DE2. A representação em *Register Transfer Level* (RTL) do sub-bloco de binarização onde as modificações propostas foram implementadas pode ser vista no Anexo A.1 que mostra de forma detalhada as modificações geradas pela inserção das entradas de seleção e pela implementação das condições baseadas nas componentes H e S do espaço HSV. Nessa representação RTL do circuito, tais modificações são vistas na forma de acréscimo de comparadores e multiplexadores.

3.3 Erosão e Filtro de Média Móvel

O fato de o resultado do cálculo do centro geométrico ser diretamente relacionado com a determinação da distância frontal ao objeto colorido de interesse, como pode-se ver na Eq. (2.6), torna a precisão e a confiabilidade desse parâmetro essencial na arquitetura de *hardware* proposta nesse trabalho. Como forma de aperfeiçoar a saída da arquitetura de segmentação por cores, foram usados dois tipos de filtros no processo de computar o centro do objeto colorido: o filtro de erosão e o filtro de média móvel.

O filtro de erosão é mais um módulo herdado das arquiteturas anteriores e é possível vê-lo instanciado no sistema de segmentação por cores apresentado na Fig. (23). A erosão é definida como uma operação básica em processamento digital de imagens e pode encolher ou eliminar objetos em um imagem digital dependendo do tamanho do elemento estruturante utilizado(GONZALEZ; WOODS, 2008). Elemento estruturante é basicamente uma máscara espacial que transcorre toda a imagem. No código em questão optou-se por um elemento estruturante de dimensões 3x3. O resultado proveniente desse algoritmo é que regiões menores que o elemento estruturante identificadas como detecção do objeto colorido são então desconsideradas. Isso, na prática, visa à eliminação do ruído

de fundo causado muitas vezes por variações na iluminação do ambiente e que podem comprometer a acurácia do algoritmo do cálculo do centro geométrico e posteriormente do cálculo da distância frontal.

Enquanto o filtro de erosão era implementado como uma máscara que percorria toda a imagem para eliminar ruídos, o filtro de média móvel é usado apenas nas coordenadas do centro geométrico calculado.

Durante a fase de testes da arquitetura de segmentação de cores notou-se que, de forma intermitente e devido a alguma variação da iluminação no ambiente ou quaisquer outros fatores dos locais de teste, a medição do centro geométrico apresentava um valor notadamente errado e depois voltava aos valores que condiziam com o centro da imagem segmentada. Uma forma de eliminar, ou suavizar, essa medição completamente fora do padrão foi a implementação do filtro de média móvel.

O seu funcionamento é simples: o valor do centro geométrico ao invés de ser a saída do cálculo instantâneo, será agora a média da medição atual com as medições passadas. A fórmula da média móvel simples é generalizada pela Eq. (3.2). Nessa equação fica claro que o valor atual da saída do filtro, $y[n]$, é dependente do valor atual da entrada, $x[n]$, como também de uma quantidade definida de valores anteriores.

$$y[n] = \frac{\sum_{k=0}^{N-1} x[n-k]}{N} \tag{3.2}$$

O número de amostras escolhidas para a implementação desta média foram 16, uma vez que multiplicações e divisões com números que são múltiplos de 2 possuem uma implementação mais simples em *hardware* e devido ao resultado satisfatório do filtro observado de forma empírica. Dessa forma, uma medição completamente equivocada será amenizada por 15 medições satisfatórias. O módulo de segmentação por cores acrescida do filtro de média móvel pode ser visualizada na Fig. (24). A arquitetura de *hardware* do filtro de média móvel gerada pela ferramenta *Quartus* a partir da descrição em Verilog é apresentada e comentada no Anexo A.2.

3.4 Arquitetura de Segmentação por Cores Duplicada

Após a validação da arquitetura de segmentação por cores, inserção da funcionalidade extra de escolha da cor a ser identificada pelo sistema e implementação de filtros que buscam agregar confiabilidade às medições, o próximo passo é duplicar a arquitetura atual a fim de se adicionar uma segunda câmera, podendo então gerar duas visões distintas do objeto colorido. A partir daí, a Eq. (2.6) pode ser usada para o cálculo da distância frontal entre o arranjo de visão estéreo e o objeto de interesse.

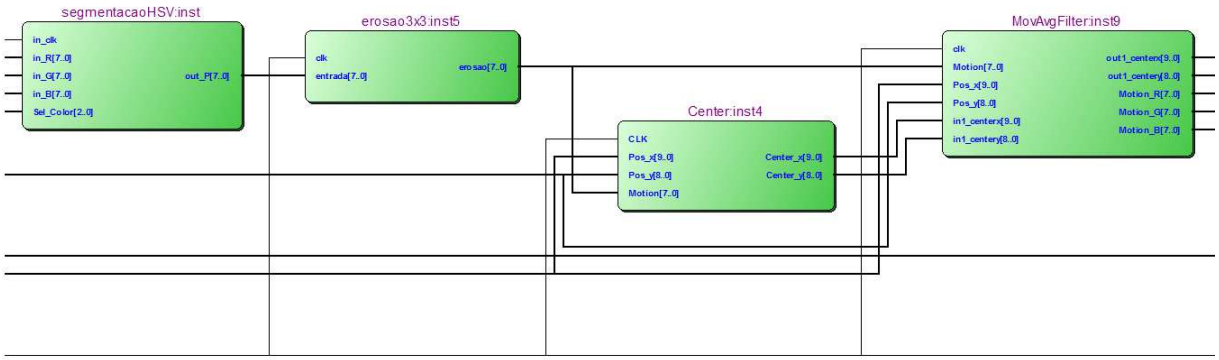


Figura 24 – Arquitetura de segmentação por cores e cálculo do centro geométrico adicionada do filtro de média móvel.

Como já mencionado, a capacidade de armazenamento do kit de desenvolvimento até então usado não suportaria a duplicação da arquitetura de aquisição de imagens, segmentação e cálculo do centro geométrico desenvolvida até o momento. A solução foi então utilizar um kit bastante similar, a placa de DE2-70, cujas diferenças literalmente se limitam a uma maior quantidade de memória e um FPGA com maior capacidade de recursos de *hardware*.

Apesar de que em uma primeira análise, o processo de se duplicar uma arquitetura seja simplesmente instanciar uma cópia de todos os módulos já existentes no projeto, a duplicação exige um conhecimento bom de todos os sinais que fazem parte do topo do projeto e a criação de novos sinais, quando necessário. Outra dificuldade é a existência de módulos que usavam o *display* LCD TRDB_LTM e seus sinais associados, a eliminação de todos eles sem critério pode excluir algum bloco que interfira também na aquisição das imagens. Vale ressaltar que essa etapa foi feita de modo concorrente com a mudança de plataformas e, por mais parecida que estas sejam, foram necessárias modificações importantes. Podem-se citar a mudança nas configurações de projeto do Quartus II que indicam o dispositivo ao qual o fluxo de projeto será desenvolvido e também a atribuição de pinos do FPGA, que relacionam os pinos da placa aos sinais declarados no módulo de topo.

Um dos prejuízos de se duplicar a arquitetura é que sem o *display* LCD TRDB_LTM a verificação da correta funcionalidade fica comprometida. Assim, ainda nessa etapa foi necessário transpor essa barreira para conseguir checar a informação dos centros geométricos calculados (agora há uma informação desse tipo para cada câmera) e, posteriormente, o cálculo da distância frontal em si. As soluções encontradas seguem duas linhas diferentes, a primeira é o uso dos LEDs de sete segmentos existentes no kit de desenvolvimento DE2-70 para visualizar os dados e a segunda é a utilização de um processador de *software* simples cuja tarefa seria basicamente enviar esses dados via porta RS-232 da placa podendo então ser lidos com um uso de um *software* de terminal simples pelo computador. Ambas as opções foram implementadas e usadas de acordo com o nível em que o projeto

estava.

3.4.1 Processador de Software NIOS II

Assim que o projeto deixou de usar o *display* LCD TRDB_LTM, o uso do processador NIOS II foi verificado como uma boa opção. Através das ferramentas da Altera *Qsys* e NIOS II SBT é possível definir a estrutura de *hardware* e programar a aplicação que esse processador executará, respectivamente. Uma vez ambientado com as ferramentas (existem diversos tutoriais que descrevem o fluxo de desenvolvimento do processador NIOS II passo a passo através do uso das ferramentas citadas acima), a modificação do *hardware* e do *software* rodando no processador é bem simples através dos recursos da ISE (*Integrated Software Environment*) da Altera.

Nesse ponto do projeto o NIOS II embarcado teve a única funcionalidade de receber os dados do centro geométrico e enviá-los pela saída serial da placa a fim de se verificar se a arquitetura duplicada funcionava da forma desejada. A Fig. (25 (a)) ilustra o bloco referente ao NIOS II instanciado nesta versão do projeto. As entradas e saídas são simplesmente o relógio do sistema, as coordenadas do centro geométrico, os fios para recebimento e envio de dados de forma serial, e uma saída de uso genérico conectada aos LEDs com o intuito de mostrar que o código instanciado no processador de *software* estava em execução.

Adiantando as implementações seguintes, o processador NIOS II teve sua estrutura de *hardware* alterada para receber o valor obtido pelo módulo de cálculo da distância frontal ao invés de receber as coordenadas do centro geométrico. Apesar disso, nessa nova estrutura pouco foi modificado no processador: as entradas de I/O de 10 e 9 *bits* foram trocadas por uma única entrada de 16 *bits*, devido à diferença de tamanho entre a informação da distância frontal e do centro geométrico; e as entradas de controle *start* e *stop* que auxiliam o fluxo da implementação final foram adicionadas (Fig. 25 (b)). O papel dessas novas entradas ficará mais claro quando for discutida a integração da arquitetura de *hardware* com o plataforma robótica.

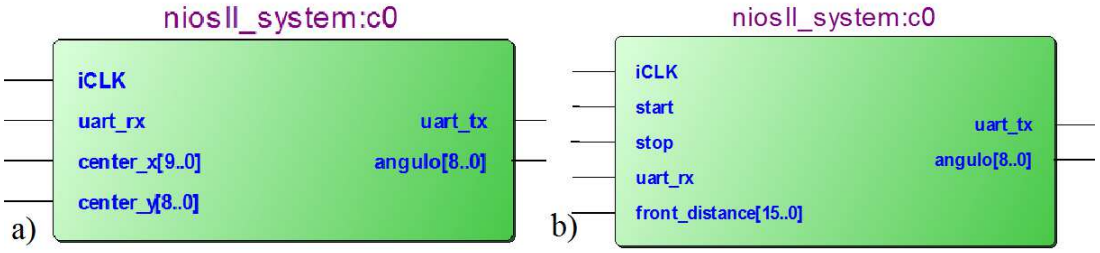


Figura 25 – Representação dos processadores NIOS II utilizados a) para validação da arquitetura duplicada e b) para controle da plataforma robótica no versão final.

Vale ressaltar também que da mesma forma que a estrutura de *hardware* foi alterada para atender aos requisitos do projeto, o programa executado pelo NIOS II também pôde ser facilmente modificado para atender às novas funcionalidades. De forma sucinta, o NIOS II da Fig. (25 (a)) basicamente recebe dados e os envia ao computador, enquanto que a informação recebida pelo processador da Fig. (25 (b)) utiliza o dado advindo do sistema para decidir se move o robô para frente ou para trás, assim nesse último caso, os dados enviados vão diretamente para o robô e não mais para o computador.

3.5 Arquitetura de Cálculo da Distância Frontal

Com a duplicação da arquitetura de segmentação concluída e seu funcionamento verificado pelos dados recebidos no terminal do computador, o código chega à fase que deixa de ter duas vertentes funcionando concorrentemente para seguir um fluxo único. Neste ponto, com informações de duas fontes distintas de um mesmo cenário, é possível determinar a distância frontal das câmeras ao objeto colorido detectado. Para isso, usa-se a semelhança de triângulos gerada pela configuração descrita na Seção 2.3.3 e a formulação matemática resultante, apresentada na Eq. (2.6).

A implementação dessa fórmula foi dividida em dois módulos consecutivos no fluxo de dados. O primeiro é o responsável pela determinação do valor da disparidade. O conceito de disparidade nada mais é do que a diferença de valores entre as coordenadas horizontais do centro geométrico calculado nas duas câmeras. De forma simples, essa medida pode ser obtida através da distância lateral esquerda subtraída da distância lateral direita (ESTEVES; FEITOSA; FERNANDES, 2012). Para o nosso caso, onde a origem do sistema de coordenadas começa no canto superior esquerdo, essas medidas nada mais são que a coordenada horizontal do centro geométrico da imagem esquerda menos a coordenada horizontal do centro geométrico da imagem direita.

O segundo módulo realiza o resto da operação para a determinação do cálculo da distância frontal. Como os demais valores envolvidos na fórmula são constantes para um determinado arranjo estereoscópico e para um modelo de câmera específico, evitou-se cálculos desnecessários codificando esse valor como uma constante no código. Essa constante engloba o valor da distância focal da câmera usada, a distância fixa entre as duas câmeras que formam o arranjo de visão estéreo e também a relação que um *pixel* mantém para a distância real. A arquitetura de *hardware* referente ao cálculo da distância frontal é ilustrada em RTL e explicada no Anexo A.3.

Tanto a distância focal como a relação de *pixels* podem ser obtidas por documentos fornecidos pelo fabricante da câmera usada. O outro fator é um parâmetro mais flexível e é determinado pelo suporte de visão estéreo, a estrutura utilizada para fixar as duas câmeras nesse trabalho usou papelão rígido e é apresentada na Fig. (26). Para a câmera

digital TRDB_D5M usada a distância focal padrão é de $7,12mm$ (KIM et al., 2014). O tamanho do *pixel* é de $2,2\mu m$ x $2,2\mu m$ (TERASIC, 2008). A distância entre as câmeras, medida com régua simples, foi verificado como sendo de $11,7cm$.

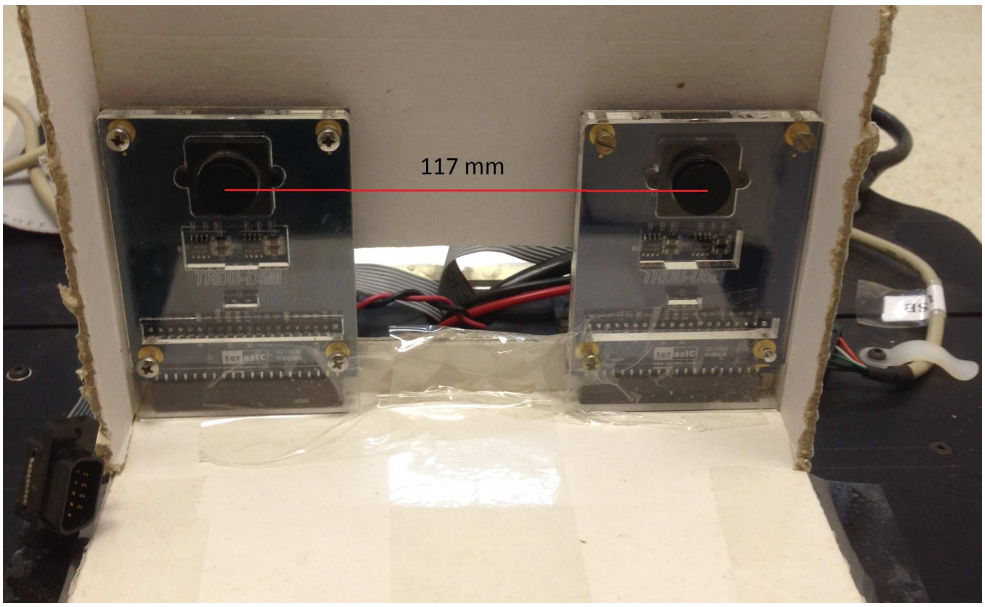


Figura 26 – Estrutura de suporte usada para o arranjo de visão estéreo.

3.6 Integração com a Plataforma Robótica

A parte final da implementação do projeto foi atingida pela integração da arquitetura de *hardware* desenvolvida com a plataforma robótica móvel. Nesse ponto, todos os módulos de *hardware* foram implementados e o acoplamento com o *Pioneer 3-AT* foi realizado via *software* usando como parâmetro a saída do cálculo da distância frontal.

O controle do robô foi desenvolvido para rodar no processador embarcado NIOS II apresentado anteriormente na Fig. (25 (b)). Apesar de a plataforma robótica possuir diversos sonares responsáveis pela localização e movimentação do robô, a aplicação implementada usou apenas a resposta da distância obtida em *hardware* para decidir a ação a ser realizada pelo robô. Vale ressaltar ainda que o envio de comandos para o microcontrolador que possui o *firmware* responsável por realizar as ações desejadas foi feito diretamente do FPGA do kit de desenvolvimento DE2-70 sem utilizar a ferramenta de desenvolvimento SDK que acompanha e auxilia a criação de aplicações para o robô.

A aplicação desenvolvida em *software* apresenta um funcionamento simples de controle do movimento do robô e tem seu fluxograma apresentado na Fig. (27). Como é visto na imagem, foram usados dois sinais de controle para auxiliar as operações, os sinais de *start* e *stop*, implementados usando duas das quatro chaves *push-button* presentes na

placa DE2-70. Tais sinais permitiram à aplicação estar em funcionamento como também em estado ocioso, o que dá flexibilidade para os testes na plataforma robótica.

O bloco de inicialização do robô envia sequências de caracteres para o estabelecimento da comunicação com o *firmware* ARCOS do processador do robô e também liga os motores e define as velocidades máximas para translação e rotação, parâmetros necessários na locomoção do robô. O bloco de finalização do robô, de forma análoga, encerra a comunicação existente e automaticamente desabilita as funções configuradas, como os motores e sonares. Assim, os parâmetros definidos retornam aos valores padrões armazenados na memória FLASH do *firmware*.

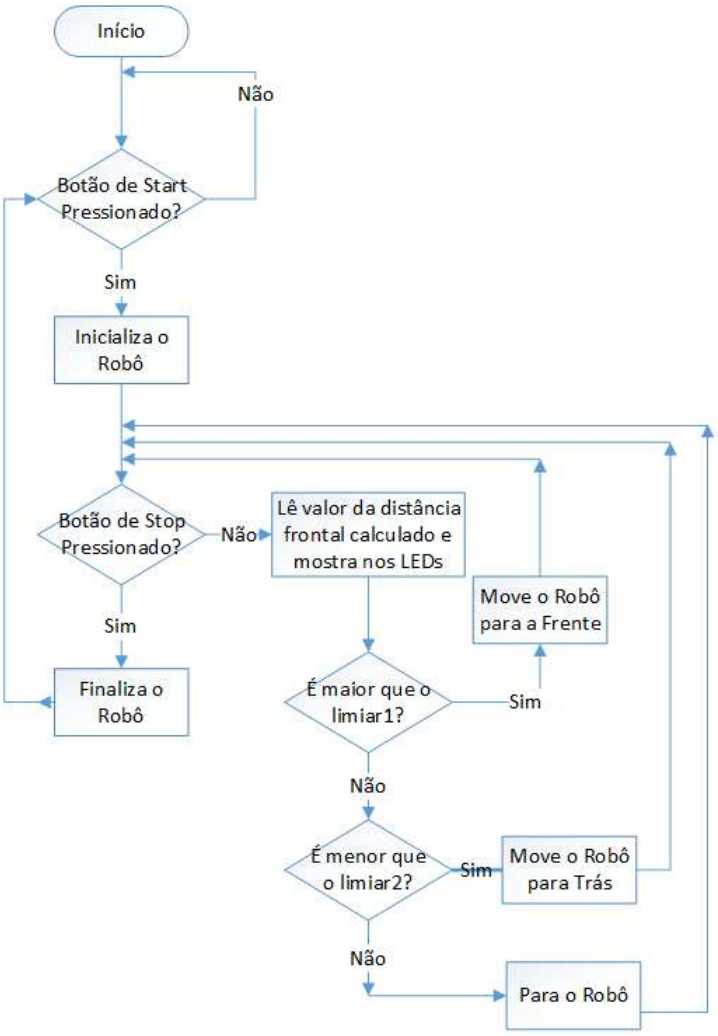


Figura 27 – Fluxograma da aplicação de *software* para o controle do movimento do robô.

Após iniciada a aplicação, os movimentos do robô são realizados de uma maneira simples através do teste de duas condições: limiar1 e limiar2 (Fig. 27). Como essa aplicação não possui um controle robusto como o encontrado em sistemas de controle modernos tais como o modelo PID, caso se definisse apenas um limiar (a distância ao qual o robô deveria permanecer ao objeto de interesse), o robô apresentaria um comportamento oscilatório

instável, ou seja, realizaria movimentos repetitivos para trás e para frente em torno da distância definida. Esse tipo de implementação permite realizar os testes na plataforma robótica de forma satisfatória à custa de um erro definido pela diferença de valores dos limiares 1 e 2. Assim, o controle implementado é do tipo *ON/OFF*, o qual verifica se o objeto está mais distante do que o desejado (limiar1) e movimenta o robô frente, caso contrário é verificado se ele está mais próximo do que deveria (limiar2) fazendo-o então mover-se para trás, e caso nenhuma dessas condições sejam satisfeitas (o robô está no intervalo de distância entre os limiares), ordena-se que o robô pare.

É importante ressaltar que a implementação que faz a interface com a plataforma robótica (envio de caracteres responsáveis para inicializar, finalizar e movimentar o robô) foi baseada no código fornecido por um aluno do Grupo de Automação e Controle (GRACO) da Universidade de Brasília, onde a plataforma robótica *Pioneer 3-AT* é usada constantemente em projetos de graduação, mestrado e doutorado dos integrantes do grupo. Vale enfatizar ainda que o envio de um comando, feito pela transmissão de uma sequência de caracteres via porta serial, não foi codificada de forma que possibilite o fácil entendimento e manutenção, uma vez que não é possível identificar quais campos e nem de que forma o *stream* de *bytes* enviados deve ser modificado para mudar os parâmetros de movimento do robô.

Como já mencionado nesse documento, a aplicação desenvolvida para o controle do robô foi executada em um ambiente controlado e com propósito acadêmico. Além disso, foram definidas condições simplificadas de rastreamento:

- Foi considerado que o objeto colorido de interesse já estará no campo de visão da plataforma robótica, ou seja, o robô não precisará fazer uma varredura no ambiente a fim de detectar em que posição da cena está o objeto.
- O robô sempre andará em linha reta, em outras palavras, os testes foram feitos para um objeto que se movimente apenas para frente e para trás em relação ao sistema de visão estéreo. Este não será capaz de acompanhar alvos que se desloquem de outra forma.

Tais condições são relevantes uma vez que a aplicação do sistema de visão estéreo na plataforma robótica foi feita com o intuito de validar o sistema desenvolvido. Formas de vencer as condições descritas acima se dão com o uso de técnicas de rastreamento em robótica móvel, o que não faz parte do escopo deste projeto.

Concluída a integração da arquitetura de *hardware* com a plataforma robótica, todos os objetivos deste trabalho foram alcançados. O sistema de cálculo da distância frontal foi implementado e seu resultado faz o controle do movimento da plataforma robótica através da aplicação de *software* rodando no processador embarcado NIOS II. A

Fig. (28) apresenta o diagrama de blocos do sistema completo resultado deste trabalho. Na imagem em questão os blocos denominados "Sistema 1" se referem à arquitetura de segmentação por cor modificada que foi então duplicada para as entradas das câmeras posicionadas em um arranjo estereoscópico. As saídas u e u' desses blocos se referem às coordenadas horizontais do centro geométrico de ambas as imagens adquiridas das câmeras esquerda e direita. Os blocos seguintes são as implementações realizadas exclusivamente por este trabalho, os quais calculam a distância entre a configuração para visão estéreo e o objeto colorido de interesse. Assim, o resultado é apresentado nos LEDs de sete segmentos do kit de desenvolvimento DE2-70 ao mesmo tempo em que serve como entrada para o processador de *software* NIOS II que faz o controle do robô. A aplicação por sua vez envia os comandos de movimento para o *firmware* ARCOS do microcontrolador do *Pioneer 3-AT* que por fim aciona os motores e permite a locomoção do robô da maneira desejada.

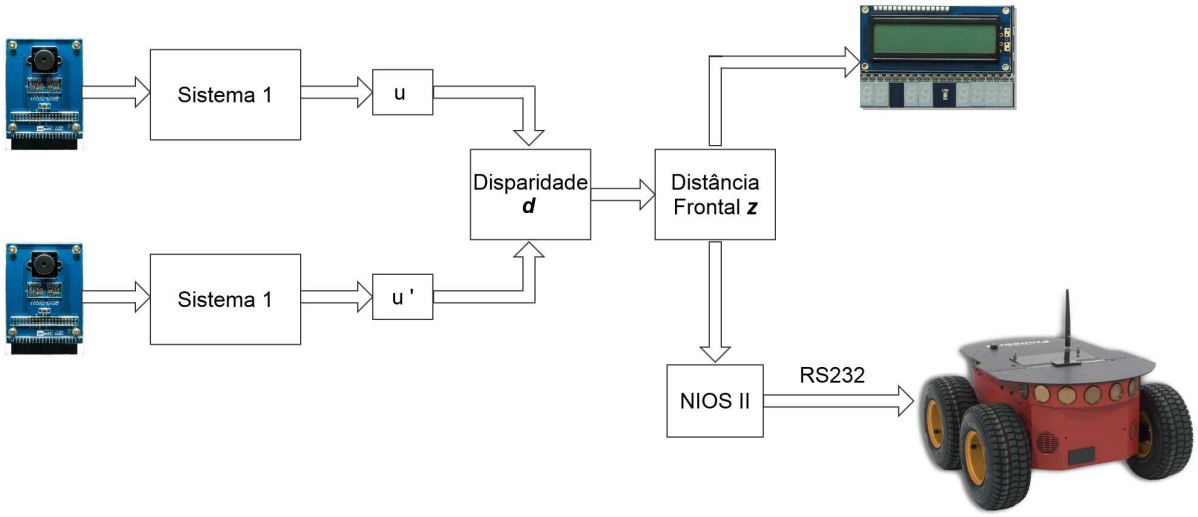


Figura 28 – Arquitetura de *Hardware* dedicada proposta no presente trabalho.

4 Resultados

4.1 Resultados de Síntese

A partir da compilação do projeto desenvolvido na ferramenta *Quartus II* são gerados diversos relatórios relativos às características da arquitetura implementada. Para cada etapa do fluxo de projeto executada ao se compilar o projeto é gerado um conjunto de relatórios contendo informações relevantes a essa etapa. Uma compilação completa na ferramenta da Altera é formada pelas seguintes etapas: análise e síntese, posicionamento e roteamento, geração dos arquivos de programação, análise temporal e geração de *netlist*.

Dentre as informações geradas a partir da síntese lógica é possível destacar a determinação da quantidade de recursos do FPGA utilizada pela arquitetura completa, como também os recursos utilizados por cada componente desta. A Tabela (3) mostra a quantidade de células lógicas (LC – *Logic Cells*) usadas tanto para implementação de funções combinacionais como também para o armazenamento de dados nos registradores, a quantidade de *bits* de memória usados e o número de blocos DSP (*Digital Signal Processor*), acompanhados de suas respectivas porcentagens em relação ao total disponibilizado pelo FPGA. A Tabela (3) mostra os recursos utilizados pelo bloco de segmentação por cores modificado, pelo filtro de erosão, pelo módulo que calcula o centro geométrico da imagem segmentada herdado de (FERREIRA, 2012), pelo filtro de média móvel, pelo conjunto dos módulos anteriores como ilustrado na Fig. (24) e referenciado na tabela como "Sistema1", pelo bloco que calcula a disparidade entre as coordenadas horizontais do centro geométrico, pelo módulo que faz a operação do cálculo da distância frontal, pelo processador de *software* NIOS II e também pelo sistema global, que incorpora módulos de aquisiçao, sincronismo e controle dos *pixels* da imagem.

Além das informações relativas à quantidade de recursos utilizados, é possível obter medidas de tempo do circuito através da ferramenta *TimeQuest* como os caminhos críticos no *design*, atrasos ou também definir restrições ao *clock* do sistema, entre outros . Uma boa medida da velocidade do circuito é a frequência máxima admitida ou simplesmente $fmax$, tal frequência é dependente do maior tempo em um caminho entre dois registradores que utilizam o mesmo *clock*. O sistema global possui dois *clocks*, o *clock* fornecido pelo kit DE2 e um implementado a partir deste através de um PLL (*Phase-Locked Loop*) que possuem frequência de $50MHz$ e $100MHz$, respectivamente. A ferramenta *Quartus* fornece a frequência máxima para todos os sinais de *clock* presentes no sistema, logo as frequências máximas relativas aos dois *clocks* do sistema foram $146.63MHz$ e $334.22MHz$, respectivamente. Tais medidas mostram que os valores de *clocks* utilizados atendem as restrições temporais da arquitetura implementada, ou seja, os *clocks* utilizados

Tabela 3 – Relatório da utilização de recursos do FPGA nas arquiteturas implementadas.

Arquitetura de *Hardware*	**LC - Funções combinacionais**	**LC - Registradores**	***Bits* de Memória**	**Blocos DSP**
Segmentação por Cores	531(<1%)	114(<1%)	0(0%)	0(0%)
Filtro de Erosão	46(<1%)	677(<1%)	16.256(1%)	0(0%)
Centro Geométrico	1160(2%)	74(<1%)	0(0%)	0(0%)
Filtro de Média Móvel	183(<1%)	160(<1%)	0 (0%)	0(0%)
Sistema 1	1919(3%)	1025(1%)	16.256(1%)	0(0%)
Disparidade	10(<1%)	10(<1%)	0 (0%)	0(0%)
Distância Frontal	304(<1%)	16(<1%)	0 (0%)	0(0%)
NIOS II Fig. (25)	2385(3%)	1699(2%)	285.696 (26%)	4(1%)
Sistema Global	9015(13%)	5865(9%)	433.008(38%)	4(1%)

no circuito estão abaixo do limite físico permitido, caso fossem maiores o funcionamento da implementação estaria compromotido.

4.2 Validação da Arquitetura de Segmentação por Cores

A fim de validar a arquitetura de segmentação por cores modificada com a opção de escolha da cor a ser segmentada pelo usuário, foram feitos dois tipos de arranjos para testes. O primeiro para certificar a identificação da cor inserida pelo usuário e segundo para mostrar a eficiência da detecção da cor desejada variando a distância do objeto de interesse e a iluminação do ambiente (Fig. 29).

A Figura (29 (a)) mostra como foi feito a validação da arquitetura para as seis cores primárias e secundárias projetadas. Foi impresso numa folha branca A4 um retângulo com cada uma das cores selecionadas. A câmera digital foi posicionada de forma que todas as cores fossem enquadradas e o *background* fosse composto apenas pela folha A4. Uma luminária de escritório foi direcionada para a cena a fim de uniformizar a iluminação. Dessa forma, a cor era informada através das chaves existentes na plataforma DE2 em uma codificação pré-estabelecida na arquitetura e o retângulo da cor desejada deveria aparecer destacado no *display* LCD através de *pixels* da cor vermelha e com um bloco no centro indicando o centro geométrico calculado. A Fig. (30) ilustra os resultados obtidos nesse arranjo para as seis cores projetadas.

As Figuras (30 (a), (b), (c), (d), (e) e (f)) mostram respectivamente as segmentações para as cores vermelha, verde, azul, magenta, ciano e amarela, de acordo com o

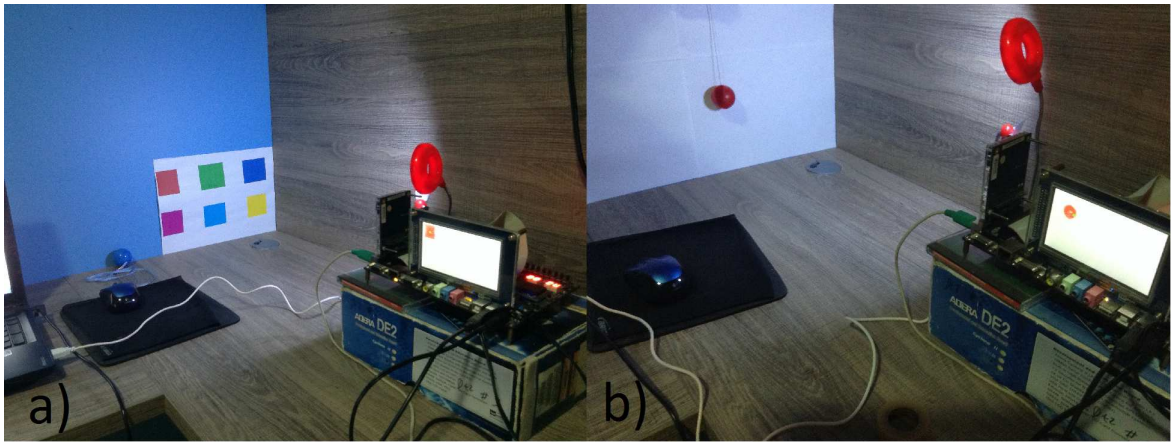


Figura 29 – Testes para validar a arquitetura de segmentação por cores (a) Detecção das cores primárias e secundárias e (b) Testes de número de acertos do centro geométrico para distância e iluminação variáveis.

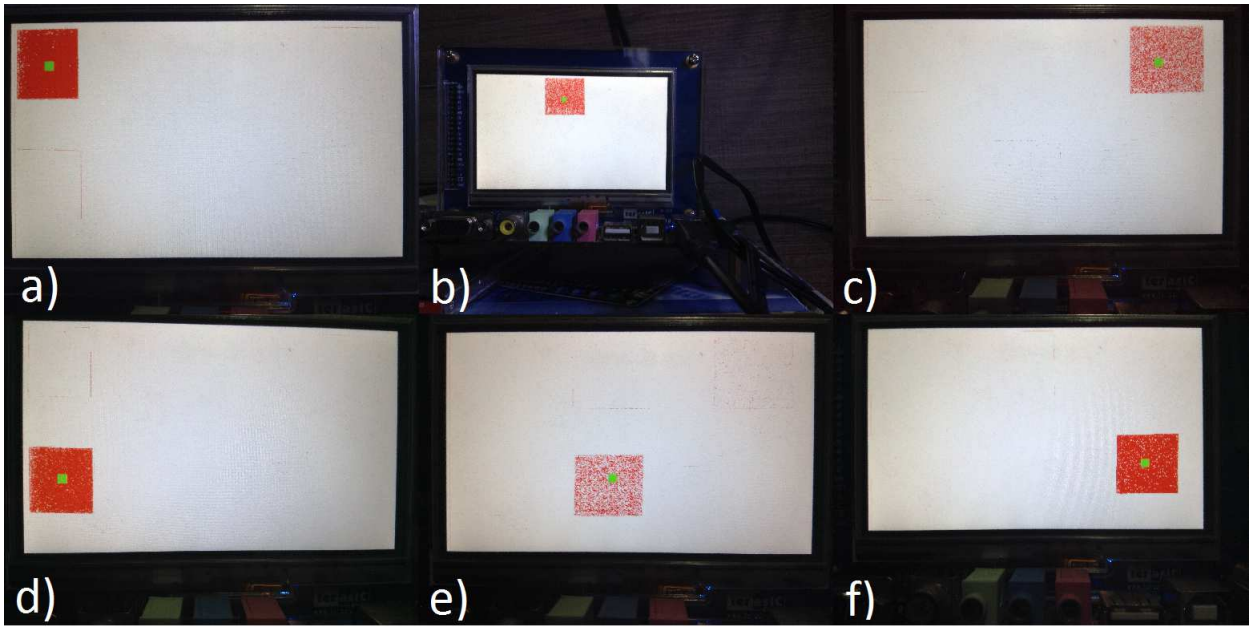


Figura 30 – Resultados apresentados no *display* LCD a para segmentação da cor (a)vermelha, (b)verde, (c) azul, (d)magenta, (e) ciano e (f) amarela.

arranjo mostrado na Fig. (29). É possível ver que a arquitetura foi capaz de identificar todas as cores projetadas apesar de que algumas cores tiveram uma quantidade de *pixels* significativa não detectados como mostra a Fig. (30 (c) e (e)), que corresponde às cores azul e ciano, respectivamente. Vale salientar também que a implementação se mostrou bastante sensível à iluminação e que em alguns casos dos resultados ilustrados foi necessário reposicionar a luminária para locais mais próximos ou mais distantes. A cor vermelha obteve o melhor resultado dentre as cores testedas usando o critério de maior densidade de *pixels* identificados analisando subjetivamente a Fig. (30), por isso, escolheu-se um objeto desta cor para a implementação do teste ilustrado na Fig. (29 (b)).

Em comparação com o teste mostrado na Figura (29 (a)), foram feitas algumas alterações para o teste da Fig. (29 (b)). Escolheu-se um objeto esférico de $4cm$ de diâmetro de uma tonalidade subjetivamente próxima à cor vermelho "pura" projetada. Foram pensadas quatro situações para a validação da correta segmentação dessa cor. No primeiro caso, a câmera ficou posicionada a uma distância de $50cm$ do fundo branco, o objeto de interesse foi colocado há uma distância de no máximo $10cm$ do fundo e a luminária esteve apontada para a cena. O segundo caso manteve todas as configurações anteriores exceto o uso da luminária. No terceiro caso a câmera estava agora a uma distância de $1m$ e a luminária fora novamente utilizada, enquanto o quarto caso manteve as configurações de distância do caso anterior, mas retirou a iluminação outra vez. Para cada um dos casos descritos, o objeto de interesse foi colocado em cinco pontos distintos do campo de visão da câmera e, para cada caso, foi verificado se o objeto fora corretamente segmentado e se a posição do centro geométrico calculado condizia com a realidade.

As vinte condições (cinco medições para cada um dos quatro casos descritos no parágrafo anterior) são apresentadas na Tab. (4) na forma de número de acertos. Nos casos em que o centro geométrico estava sobre o objeto segmentado (no centro ou nas bordas deste) foi considerado acerto e caso que o centro geométrico foi detectado como fora do objeto de interesse, foi determinado erro.

Tabela 4 – Número de acertos do centro geométrico para diferentes distâncias e iluminação.

	Centro	**Bordas**	**Fora**	**Acertos**	**Erros**
1° Caso:50 cm e boa iluminação	5	0	0	5	0
2° Caso:50 cm e má iluminação	5	0	0	5	0
3° Caso:1 m e boa iluminação	2	2	1	4	1
4° Caso:1 m e má iluminação	2	2	1	4	1
Total	14	4	2	18	2

Analisando a Tabela (4) é fácil verificar que para distâncias maiores o cálculo do centro geométrico ficou mais propício ao erro. Apesar da dependência da iluminação não ser perceptível pela Tab. (4), esta interfere negativamente no sentido do centro geométrico oscilar bastante durante os testes, este alternando entre fora e dentro da imagem segmentada para uma mesma medição, o que também não é desejável. Vale salientar que o objeto testado era muito pequeno ($4cm$ de diâmetro) e que os problemas do cálculo geométrico eram devidos principalmente à iluminação e ao tamanho do objeto, ou seja, para uma iluminação ruim pontos ruidosos eram detectados no *background* como *pixels* de interesse, o que deslocava o centro geométrico do lugar desejado e, quando o objeto ficava mais distante, esses *pixels* ficavam mais significativos uma vez que a quantidade de pontos referentes ao objeto diminuiu (o objeto estava menor na imagem).

Um exemplo de cada um dos casos testados é mostrado na Figura (31), onde as

Fig. (31 (a), (b), (c) e (d)) ilustram o primeiro, segundo, terceiro e quarto casos analisados na Tab. (4), respectivamente.

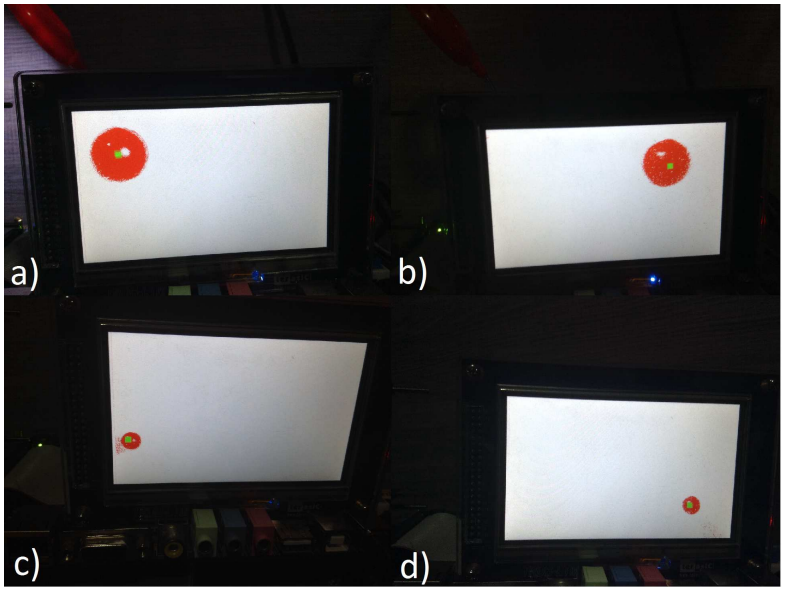


Figura 31 – Resultados do teste de número de acertos do centro geométrico apresentados no *display* LCD para (a) $50cm$ e boa iluminação, (b) $50cm$ e má iluminação, (c) $1m$ e boa iluminação e (d) $1m$ e má iluminação.

4.3 Validação da Arquitetura de Cálculo da Distância Frontal

Para validar a arquitetura de cálculo da distância frontal foi definido o seguinte teste: um retângulo vermelho de dimensões $24cm$ x $16cm$ foi fixado em uma parede de cor branca a uma altura de $37,7cm$. O arranjo de visão estéreo foi posicionado a cinco distâncias diferentes do objeto: $1m$; $1,5m$; $2m$; $2,5m$ e $3m$. Tal distância foi verificada utilizando uma trena. O arranjo estava a uma altura de $52cm$ do solo (Fig. 32).

Antes de iniciar o teste propriamente dito, era necessário utilizar os códigos produzidos em passos anteriores (os quais faziam o uso do *display* LCD TRDB_LTM) para confirmar que o objeto colorido estava totalmente incluído no campo de visão de ambas as câmeras. Após confirmado a inclusão do objeto; utilizou-se o código da arquitetura de segmentação duplicada, os módulos de cálculo da distância frontal e o processador NIOS II embarcado ilustrado na Fig. (25 (b)). Assim, em cada uma das distâncias selecionadas, a aplicação executada no NIOS II lia os valores da distância frontal e enviava via porta RS-232 os dados. Para facilitar a interpretação e a análise de erros dos dados, a informação das distâncias lidas eram apresentadas em milímetros no programa emulador de terminal *Tera Term* no computador. Usou-se o recurso do *Tera Term* de salvar as informações recebidas por comunicação serial em um arquivo. Devido à velocidade de processamento

Figura 32 – Realização dos testes da arquitetura do cálculo da distância frontal.

uma quantidade muito grande de amostras era recebida em poucos segundos. Para análise de erros das medições foram padronizados o número de 1000 amostras por medição.

Para a análise das medidas realizadas usou-se uma rotina simples implementada no *software* MATLAB que calcula a média de cada um dos grupos de amostras, o erro relativo da média das medidas em relação à distância definida com o uso da trena (Eq. 4.1) e também o desvio padrão da média (Eq. 4.2). Na Eq. (4.1) o símbolo X' se refere ao valor aproximado e enquanto o Xv referencia o valor verdadeiro. Para a Eq. (4.2), X_i é o valor de uma medida enquanto $\bar{X}$ é o valor da média.

A Tabela (5) mostra os erros associados às medições de distância frontal nos cinco comprimentos definidos.

$$Er\% = \left| \frac{X' - Xv}{Xv} \right| \times 100 \tag{4.1}$$

$$s = \sqrt{\frac{\sum_{i}^{n}(X_i - \bar{X})^2}{n}} \tag{4.2}$$

Analisando a Tabela (5) pode-se notar que as médias dos valores medidos alcançaram um grau de precisão muito bom, confirmado quando se incorpora os valores percentuais dessa análise, obtendo erros relativos próximos, ou menores, que 1% em quase todas as distâncias. Entretanto quando se verifica os desvios padrões das medidas nota-se que à medida que a distância aumenta, a dispersão das medições em torno da média aumenta. Tal resultado é esperado tendo em vista que para distâncias maiores a disparidade entre

Tabela 5 – Análise de erros do teste da arquitetura de cálculo da distância frontal.

Distância das Medições	Média (mm)	Erro Relativo (%)	Desvio Padrão (mm)
1 m	1054	5,36%	4,8727
1,5 m	1519	1,26%	9,6052
2 m	2008	0,42%	21,5886
2,5 m	2514	0,57%	43,3762
3 m	3030	1,01%	84,4042

os centros geométricos diminui, causando então que cada *pixel* tenha um peso maior no cálculo da distância.

4.4 Validação da integração com a Plataforma Robótica *Pioneer 3-AT*

Para a validação da arquitetura de cálculo da distância frontal controlando o robô *Pioneer 3-AT* utilizou-se uma abordagem semelhando aos testes realizados na seção anterior. Utilizando um retângulo vermelho de mesmas dimensões colado em uma parede e utilizando uma trena como referência de medição das distâncias, fixou-se a estrutura de suporte das câmeras na parte superior da plataforma robótica juntamente com o kit de desenvolvimento DE2-70 (33).

A abordagem escolhida inicialmente foi, para cada uma das distâncias definidas na seção anterior, posicionar o robô um pouco mais próximo e depois um pouco mais distante da distância escolhida e verificar o comportamento do robô quando é alcançada a distância desejada. Em um primeiro momento, utilizou-se uma versão da aplicação de *software* mais simples do que aquela apresentada no fluxograma da Fig. (27): nesse algoritmo mais simples, implementou-se apenas um limiar (a distância desejada) e o robô basicamente se moveria para frente e para trás de acordo com o resultado da comparação da distância calculada com esse limiar.

Nessa configuração, o robô se comportava de maneira oscilatória em torno da distância determinada, como esperado, porém oscilava mais do que se esperaria nessa situação. Esse comportamento é explicado tendo como referência a Fig. (21) apresentada na Seção 3.1.7, a qual descreve a maneira de se locomover do robô. A plataforma robótica *Pioneer 3-AT* apresenta um perfil de velocidades trapezoidal, ou seja, tanto para se mover quanto para brecar, respeita a inclinação da aceleração e desaceleração mostrada na figura. Como explicado na Seção 3.6 que explana sobre a aplicação de *software*, os comandos enviados ao robô foram implementados de forma que não se tem o conhecimento sobre os parâmetros definidos e assim, não é trivial modificá-los, tornando muito complexo a alteração das acelerações e desacelerações definidas e que causam o efeito oscilatório

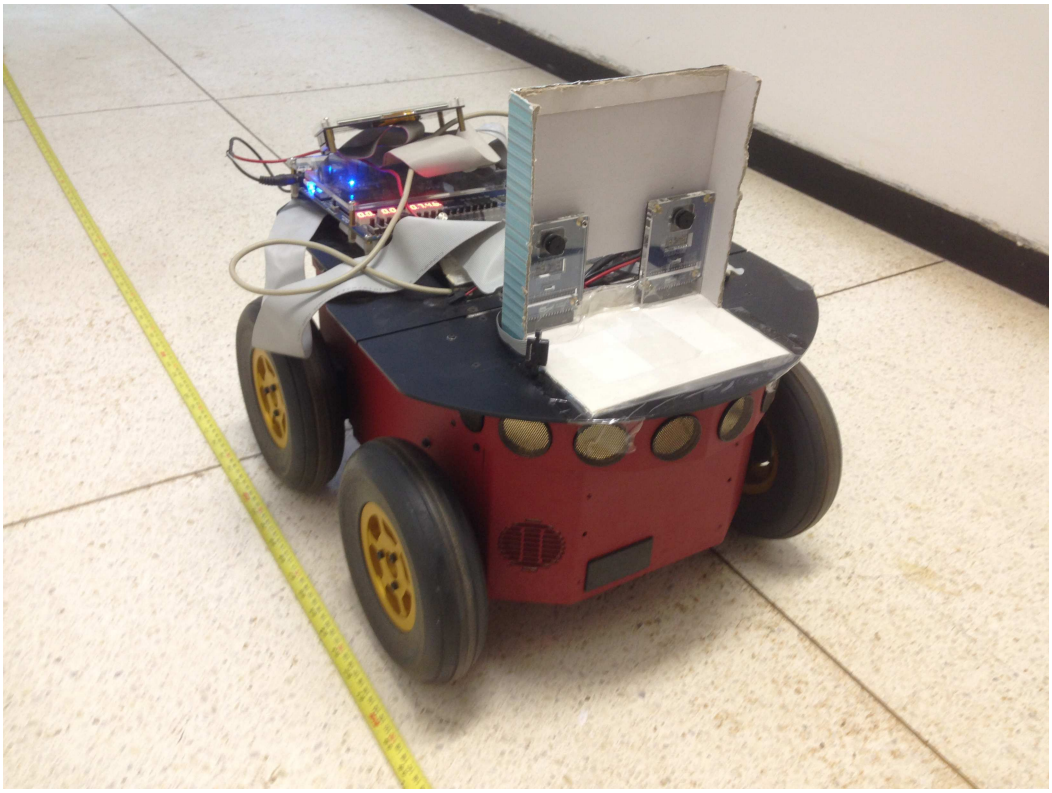


Figura 33 – Realização dos testes da arquitetura desenvolvida integrada à plataforma robótica.

indesejado.

Como forma de continuar os testes com o robô e tendo em mente que o foco principal do projeto é a validação da arquitetura de *hardware* controlando a plataforma robótica *Pioneer 3-AT* e não o domínio do robô, utilizou-se a abordagem descrita no fluxograma (27) que, ao custo de perda de acurácia, utiliza dois limiares e envia também comandos de parada ao robô. Com essa nova forma de controle, foi possível realizar os testes na forma desejada. Apesar de apresentar pequena oscilação em torno da distância definida, os comandos de parada brecavam rapidamente a oscilação e o robô estabilizava em uma distância próxima à definida. A Tabela (6) apresenta os limiares superior e inferior definidos para cada uma das distâncias testadas assim como a distância a qual o robô se estabilizou. Os índices 'F' e 'T' significam "frente" e "trás", respectivamente, e se referem à posição de partida do robô em relação à posição desejada.

A partir das medições pode-se ver que a arquitetura de *hardware* é capaz de controlar os movimentos do robô de acordo com a distância estabelecida. As medições da Tabela (6) mostram que para a distância de $1m$ o robô apresentou ótimos resultados, bastante próximos à distância desejada, mas que para distâncias maiores um erro de posicionamento foi observado. Uma das causas pode ser determinada a partir da Tabela (5) que apresenta uma análise de erros das medidas observadas no teste do módulo da distância frontal. Da Tabela (5) pode-se ver que apesar da média das medidas convergirem para

Tabela 6 – Resultados dos testes da arquitetura de cálculo da distância frontal integrada à plataforma robótica *Pioneer 3-AT*

Distância das Medições	**Limiar (mm) Inferior**	**Limiar (mm) Superior**	**Distância (cm)** $Final_F$	**Distância (cm)** $Final_T$
1 m	950	1050	100	101
1,5 m	1450	1550	160	157
2 m	1930	2080	215	209
2,5 m	2400	2600	266	263
3 m	2900	3100	310	317

a distância desejada, há uma dispersão (observada pelo desvio padrão calculado) dessas à medida que a distância ao objeto colorido aumenta. Isso é explicado a partir da Eq. (2.6) apresentada. Para distâncias maiores, o valor da disparidade é menor. Dessa forma, cada *pixel* tem maior relevância na distância frontal calculada, logo, a resolução diminui causando medições com valores mais esparsos entre si. Voltando à plataforma robótica, esses valores mais espaçados foram as causas para se implementar limiares mais distantes na aplicação de controle do robô (Tabela 6) e que influenciam na acurácia da medida. Não só a mudança dos limiares, mas também as próprias medições com resolução menor interferem na precisão do controle do robô para a distância desejada, culminando nas diferenças entre os valores reais e os valores obtidos para distâncias maiores, de acordo com a Tabela (6).

Como último teste para a integração da plataforma robótica com a arquitetura de *hardware* para cálculo da distância frontal, desejou-se verificar o comportamento do controle do robô para um objeto móvel. Para isso utilizou-se um retângulo vermelho de dimensões $30,3cm$ x $22,4cm$ fixado em uma régua transparente de $60cm$. Assim, para a distância de $1m$ que apresentou os melhores resultados segundo a Tabela (6), aproximou-se e afastou-se o arranjo descrito do campo de visão do robô diversas vezes (Fig. 34).

Esse último teste possuiu um cunho essencialmente qualitativo, uma vez que o objetivo foi verificar o comportamento do robô para um objeto móvel. Apenas para mostrar que o robô além de se locomover apropriadamente também se manteve próximo à distância desejada. Foram tomadas três medidas entre o robô e o objeto colorido móvel (Tabela 7). Para essas medidas não foram realizados análise de erros uma vez que foge do escopo desse último teste.

Tabela 7 – Medições realizadas no teste da arquitetura integrada ao robô para um objeto colorido móvel

Posição do Objeto (cm)	**Posição do Robô (cm)**	**Distância (cm)**
25	121	96
56	158	98
76	181	95

Figura 34 – Realização dos testes da arquitetura do cálculo da distância frontal para um objeto colorido móvel.

Vale salientar que para todos os testes realizados com a arquitetura de *hardware* integrada ao robô foram realizadas filmagens para documentar o processo.

5 Conclusões e Trabalhos Futuros

5.1 Conclusões

O uso da plataforma FPGA é especialmente interessante para a implementação de arquiteturas de processamento de imagens, uma vez que estas apresentam um alto grau de paralelismo intrínseco tanto no escopo espacial, lógico e temporal. Arquiteturas de *hardware* fazem uso desse paralelismo para acelerar algoritmos de processamento de imagens melhorando o desempenho computacional da aplicação em comparação com os mesmos algoritmos implementados em processadores de *software*, que possuem como restrição o uso de arquiteturas tipo *Von Neumann*. Juntamente com isso, os FPGAs modernos estão em crescente avanço tecnológico fornecendo cada vez mais uma maior quantidade de recursos lógicos, *clocks* mais rápidos e blocos ASIC embarcados aumentando ainda mais o desempenho dessa plataforma, tornando-o ideal para sistemas de visão em tempo real.

Uma plataforma robótica provida de mobilidade e um sistema de visão está apta a uma grande variedade de aplicações tais como detecção e rastreamento de objetos. Assim, o desenvolvimento de um arranjo de visão estéreo em *hardware* reconfigurável e aplicado em robótica móvel possibilitou a implementação de um sistema que guia e determina as ações de uma plataforma robótica de acordo com a distância frontal a um objeto de interesse. Aliando o sistema de visão com uma arquitetura de *hardware* de segmentação por cores, o sistema desenvolvido foi capaz de se controlar a plataforma robótica baseado na distância a objetos de uma cor específica.

A síntese lógica da arquitetura de segmentação por cores gerada mostrou um uso de 3% de células lógicas, 1% da memória disponível e um bloco DSP dos recursos disponíveis na plataforma FPGA DE2-70. As arquiteturas de cálculo da disparidade e determinação da distância frontal ocuparam juntas menos 1% de células lógicas. O uso do processador de *software* embarcado na arquitetura fez uso de 3% de células lógicas, aproximadamente 26% dos *bits* de memória e 4 blocos DSPs. Por fim o sistema global, abrangendo todos os módulos anteriores, utilizou 13% de células lógicas, 38% da memória interna do FPGA e um total de 4 DSPs. Ainda, o sistema total apresentou uma frequência máxima de operação de $146,63MHz$.

Os testes da arquitetura de segmentação de imagens por cores mostraram uma efetividade do sistema atingindo uma eficiência de 90% segundo o número de acertos do cálculo do centro geométrico do objeto segmentado. Os testes da arquitetura de cálculo da distância frontal obtiveram resultados bastante satisfatórios com a média das medições bem próxima à medida real e alcançando erros relativos em torno, e até inferior, a 1%.

Os testes realizados mostraram que a arquitetura de *hardware* foi capaz de controlar o movimento do robô da forma desejada. Os resultados para a distâncias curtas foram considerados ótimos enquanto os resultados para distâncias mais longas perderam um pouco a precisão, mas ainda foram subjetivamente próximos às distâncias definidas. A perda da precisão se deve ao fato de que as medições obtidas para distâncias mais longas têm uma resolução menor e por alterações nos limiares que exerciam o controle da plataforma robôtica.

5.2 Propostas de Trabalhos Futuros

Os projetos que queiram usar o presente trabalho como ponto de partida terão duas linhas distintas a seguir. Um segmento, que ficou claro durante a realização do projeto, é a necessidade de se implementar um método de segmentação por cores robusto que não se limite a ambientes específicos e também não tenha tanta sensibilidade à iluminação, o que se mostrou um grande problema. Algoritmos de processamento de imagens geralmente utilizam vários passos e frequentemente usam métodos redundantes para determinada tarefa, a fim de agregar robustez e confiabilidade ao processo. Assim, outras formas de validar a segmentação por cores desenvolvidas em *hardware* que funcionem concorrentemente ou que substituam a atual implementação são possíveis trabalhos a se fazer. A partir do novo modo encontrado, refazer os últimos testes e utilizar as arquiteturas desenvolvidas nesse trabalho em ambientes mais dinâmicos proporcionariam avanços significativos no projeto.

O outro ramo a se tratar para projetos futuros é a implementação de um sistema de controle adequado na movimentação do robô. Desde o início, esse trabalho teve como objetivo aplicar a arquitetura de cálculo da distância frontal desenvolvida em *hardware* para controlar uma plataforma robótica móvel da forma mais trivial possível, apenas dando comandos simples de translação para trás e para frente. Entretanto, sistemas reais necessitam de um controle mais elaborado. Esse tipo de controle é obtido através de um controlador PID (Proporcional Integral Derivativo). A implementação de uma arquitetura de *hardware* que realize este controle a partir dos resultados brutos do cálculo da distância frontal e aplique de forma satisfatória na plataforma robótica traria grandes ganhos e oportunidades para o projeto.

Referências

ADEPT MOBILEROBOTS. *Pioneer 3 Operations Manual*. New Hampshire, United States, 2010. Citado na página 52.

ADEPT TECHNOLOGY, INC. *Pioneer 3-AT*. New Hampshire, United States, 2011. Citado 2 vezes nas páginas 15 e 52.

ALTERA CORPORATION. *DE2 Development and Education Board*: User manual. [S.l.], 2012. Citado 2 vezes nas páginas 17 e 45.

BAIELY, D. G. *Design for Embedded Image Processing on FPGAs*. [S.l.]: John Wiley & Sons, 2011. Citado 8 vezes nas páginas 15, 23, 24, 38, 39, 40, 41 e 42.

BITTENCOURT, A. P. C. *Controle de Sistema de Dois Graus de Liberdade com Realimentação Visual por Meio de Segmentação por Cores em Plataforma Reconfigurável*. Dissertação (Mestrado) — Universidade de Brasília, 2012. Citado 9 vezes nas páginas 15, 25, 26, 45, 46, 47, 53, 54 e 55.

BRAUNL, T. *Embedded Robotics*. Segunda edição. [S.l.]: Springer, 2003. Citado na página 23.

DEHON, A. The density advantage of configurable computing. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 33, n. 4, p. 41–49, Abril 2000. Citado na página 42.

ESTEVES, G. R. P.; FEITOSA, M. A.; FERNANDES, B. J. T. Estereoscopia no cálculo de distância e controle de plataforma robótica. In: *SIBGRAP - Conference on Graphics, Patterns and Images*. Ouro Preto, Brasil: [s.n.], 2012. p. 65–70. Citado 3 vezes nas páginas 23, 24 e 60.

FERNANDEZ, F. et al. A methodology for reconfigurable hardware design based upon evolutionary computation. *MICPRO Microprocessors and Microsystems: Embedded Hardware Design*, v. 28, n. 7, p. 363–371, Setembro 2004. Citado na página 23.

FERREIRA, C. S. *Implementação do Algoritmo de Subtração de Fundo para Detecção de objetos em Movimento Usando Sistemas Reconfiguráveis*. Dissertação (Mestrado) — Universidade de Brasília, 2012. Citado 3 vezes nas páginas 53, 54 e 65.

GARCIA, P. et al. An overview of reconfigurable hardware in embedded systems. *EURASIP Journal on Embedded Systems*, v. 2006, n. 1, p. 1–19, Junho 2006. Citado 2 vezes nas páginas 23 e 38.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. [S.l.: s.n.], 2008. Citado 5 vezes nas páginas 15, 29, 30, 31 e 56.

HAUCK, S.; DEHON, A. *Reconfigurable Computing. The Theory and Practice of FPGA-based Computation*. [S.l.]: Morgan Kaufmann, 2008. Citado na página 39.

JOBLOVE, G. H.; GREENBERG, D. Color spaces for computer graphics. In: *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*. Nova York, Estados Unidos: [s.n.], 1978. p. 20–25. Citado na página 31.

KILTS, S. *Advanced FPGA Design - Architecture, Implementation and Optimization*. [S.l.]: John Wiley & Sons, 2007. Citado na página 42.

KIM, J. H. et al. *Robot Intelligence Technology and Aplications 2 - Results from the 2nd International Conference on Robot Intelligence Technology and Applications*. [S.l.]: Springer, 2014. Citado na página 61.

LENART, T. *Design of Reconfigurable Hardware Architectures for Real-time Applications*. 175 p. Tese (Doutorado) — Lund University, 2008. Citado na página 42.

RUS, J. *Hsl-hsv models b*. 2010. <http://commons.wikimedia.org/wiki/File:Hsl-hsv_models_b.svg>. [Online; acessado 27-05-2014]. Citado 2 vezes nas páginas 15 e 32.

SANTOS, M. C. *Revisão de Conceitos em Projeção, Homografia, Calibração de Câmera, Geometria Epipolar, Mapas de Profundidade e Varredura de Planos*. 2012. <http://www.ic.unicamp.br/~rocha/teaching/2012s1/mc949/aulas/additional-material-revision-of-concepts-homography-and-related-topics.pdf>. [Online; acessado 01-06-2014]. Citado 3 vezes nas páginas 15, 34 e 35.

SASS, R.; SCHMIDT, A. G. *Embedded Systems Design with Plataform FPGAs. Principles and Practices*. [S.l.]: Morgan Kaufmann, 2010. Citado 5 vezes nas páginas 15, 23, 40, 42 e 43.

SIEGWART, R.; NOURBAKHSH, I. R. *Introduction to Autonomous Mobile Robots*. [S.l.]: The MIT Press, 2004. Citado na página 23.

SMITH, A. R. Color gamut transform pairs. In: *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*. Nova York, Estados Unidos: [s.n.], 1978. p. 12–19. Citado 2 vezes nas páginas 24 e 33.

SONKA, M.; HLAVAC, V.; BOYLE, R. *Image Processing, Analysis, and Machine Vision*. [S.l.: s.n.], 2008. Citado 6 vezes nas páginas 15, 29, 34, 36, 37 e 38.

TERASIC TECHNOLOGIES. *LTM*: User manual. [S.l.], 2003. Citado 2 vezes nas páginas 15 e 48.

TERASIC TECHNOLOGIES. *TRDB_D5M*: 5 mega pixel digital camera development kit. [S.l.], 2008. Citado 3 vezes nas páginas 15, 48 e 61.

TERASIC TECHNOLOGIES. *DE2-70 Development and Education Board*: User manual. [S.l.], 2009. Citado 2 vezes nas páginas 15 e 47.

UDE, A. *Robot Vision*. [S.l.]: In-Teh, 2010. Citado 2 vezes nas páginas 23 e 24.

Anexos

ANEXO A – Arquiteturas de Hardware

A.1 Arquitetura do Módulo de Segmentação por Cores Modificado

A Fig. (A.1) representa a arquitetura de *hardware* gerada para uma parte do módulo de segmentação por cores, o bloco de binarização. Apesar de outras pequenas modificações ter sido feitas por todo o módulo a fim de deixá-lo funcional, esta parte de código foi a que mais recebeu alterações do presente trabalho.

Analisando as estruturas que recebem as entradas, é fácil verificar que os comparadores juntamente com as portas lógicas NAND implementam as condições para que uma determinada cor seja segmentada, conforme explicado na Seção 3.2.2. Na Fig. (A.1 nota-se que o resultado desse bloco combinacional vai para um multiplexador que por sua vez é controlado pelas entradas de seleção (chaves SW0, SW1 e SW2) do kit de desenvolvimento. Por fim, caso os valores das componentes HSV de entrada atendam às condições para a cor selecionada, um registrador de saída recebe o nível lógico 1, cor identificada, caso contrário recebe o nível lógico 0, cor não identificada.

A.2 Arquitetura do Filtro de Média Móvel

Devido ao tamanho e à repetição de blocos, a Fig. (A.2) mostra apenas a parte inicial da arquitetura de *hardware* gerada para filtro de média móvel. Os blocos em azul são registradores, responsáveis por amazenar o valor das coordenadas do centro geométrico. Os registradores no topo armazenam valores para a coordenada horizontal e os valores inferiores armazenam para a coordenada vertical. Pode-se ver que os registradores estão posicionados em cadeia, ou seja, o entrada da coordenada do centro geométrico entra no primeiro registrador enquanto a coordenada obtida a 16 ciclos passados não é mais utlizada pelo último registrador. Para os demais registradores da sequência, ocorre uma atualização com o valor armazenado pelo bloco anterior.

Os somadores na parte debaixo do circuito são os responsáveis por acumular o valor total dos registrados. No final dessa malha de registradores e somadores, é realizada a divisão do valor acumulado por 16, valor definido pela facilidade da sua implementação em *hardware*. Não é mostrado na figura, mas por ser um valor múltiplo de 2, essa divisão é realizada simplesmente pelo deslocamento dos *bits* do resultado de 4 posições. Isso porque cada deslocamento equivale a uma divisão ou multiplicação por 2 dependendo do sentido ($2^4 = 16$).

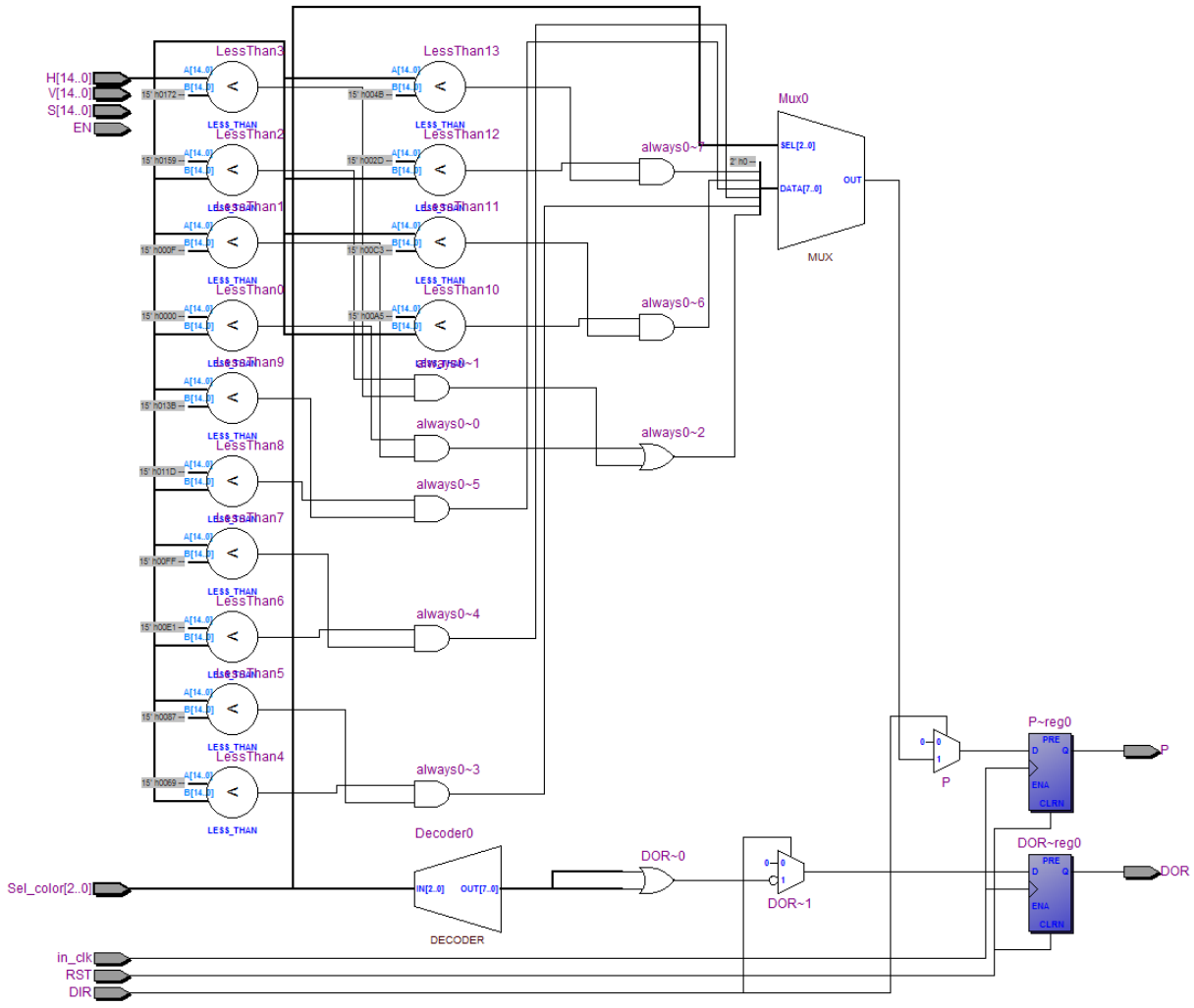


Figura 35 – Representação RTL do sub-bloco de binarização do módulo de segmentação por cores após modificação.

A.3 Arquitetura do Módulo de Cálculo da Distância Frontal

Apesar de toda a teoria desenvolvida e discutida na Seção 2.3, a implementação do cálculo da distância frontal segue uma fórmulação matemática simples e direta (2.6). Assim, uma vez duplicada e validada a arquitetura de segmentação por cores, a produção do bloco de distância frontal é básico. A arquitetura de *hardware* produzida pela descrição em Verilog se resume à Fig. (A.3), sendo que a Fig. (A.3 (a)) se refere ao módulo de cálculo da disparidade e a Fig. (A.3 (b)) representa o bloco que determina a distância frontal.

Como o conceito de disparidade indica, a disparidade entre as coordenadas horizontais do centro geométrico das câmeras é simplismente a subtração de seus respectivos valores, nesse caso o valor da câmera esquerda menos o da câmera à direita. Este valor é armazenado no registrador de saída e é então fornecido como parâmetro para o bloco do cálculo da distância frontal (Fig. A.3 (a)). Tão simples como o *hardware* anterior,

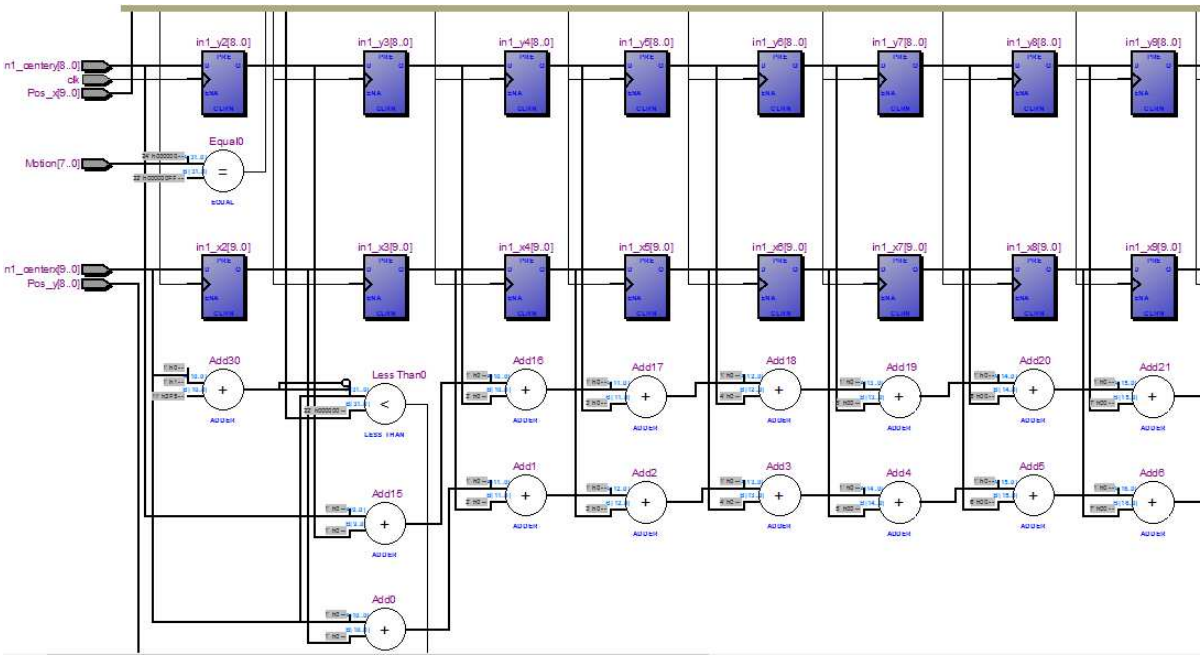


Figura 36 – Representação RTL de parte da arquitetura do filtro de média móvel.

a arquitetura de cálculo da distância frontal é apenas um divisor. Esse arquitetura foi simplificada uma vez que para uma configuração estéreo específica e câmeras com distância focal e resolução conhecidas, a Eq. (2.6) é formada quase em sua totalidade por valores constantes. Logo, para evitar processamento desnecessário, o valor resultante das operações entre constantes foi fixado como parâmetro no cálculo apresentado. Assim, o valor da distância frontal é obtido simplesmente pela divisão dessa constante pelo valor da disparidade. Nesse caso, por ser uma divisão qualquer, foi instanciado pela ferramenta de síntese um bloco IP (*Intelectual Property*) específico e otimizado para essa função sendo que o método utlizado para divisão no módulo do filtro de média móvel não é adequado.

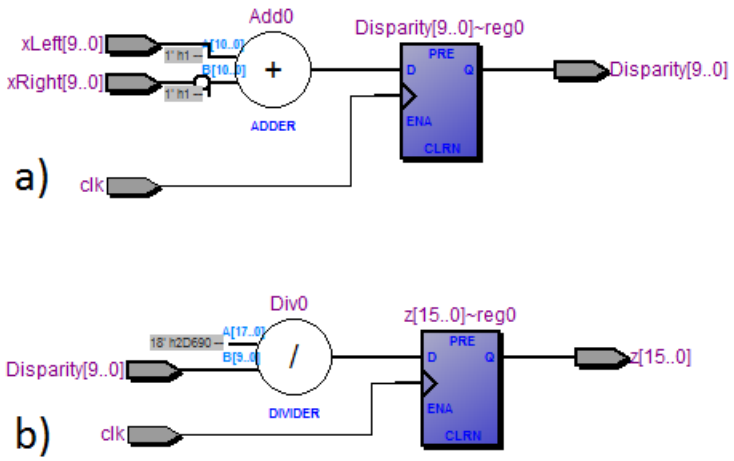


Figura 37 – Representação RTL do módulo de cálculo da distância frontal.